UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE AUTOMAÇÃO E SISTEMAS
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Francisco Silveira Burigo

Integração e Entrega Contínua para Serviços Serverless em Cloud: Um Estudo de Caso para Processamento de Dados em Data Lake para uma Empresa do Mercado de Aluguel por Temporada.

Florianópolis
2023

Francisco Silveira Burigo

Integração e Entrega Contínua para Serviços Serverless em Cloud: Um Estudo de Caso para Processamento de Dados em Data Lake para uma Empresa do Mercado de Aluguel por Temporada.

Relatório final da disciplina DAS5511 (Projeto de Fim de Curso) como Trabalho de Conclusão do Curso de Graduação em Engenharia de Controle e Automação da Universidade Federal de Santa Catarina em Florianópolis.
Orientador: Prof. Ricardo José Rabelo, Dr.

Florianópolis
2023

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Burigo, Francisco Silveira
Integração e Entrega Contínua para Serviços Serverless em Cloud: Um Estudo de Caso para Processamento de Dados em Data Lake para uma Empresa do Mercado de Aluguel por Temporada / Francisco Silveira Burigo ; orientador, Ricardo José Rabelo, 2023.
108 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Engenharia de Controle e Automação, Florianópolis, 2023.

Inclui referências.

1. Engenharia de Controle e Automação. 2. DevOps. 3. Integração Contínua. 4. Entrega Contínua. 5. Processamento de dados. I. Rabelo, Ricardo José. II. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. III. Título.

Francisco Silveira Burigo

Integração e Entrega Contínua para Serviços Serverless em Cloud: Um Estudo de Caso para Processamento de Dados em Data Lake para uma Empresa do Mercado de Aluguel por Temporada.

Esta monografia foi julgada no contexto da disciplina DAS5511 (Projeto de Fim de Curso) e aprovada em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação

Florianópolis, 11 de 07 de 2023.

Prof. Hector Bessa Silveira, Dr.
Coordenador do Curso

Banca Examinadora:

Prof. Carlos Montez, Dr.
Avaliador
UFSC/CTC/DAS

Prof. Ricardo José Rabelo, Dr.
Orientador
UFSC/CTC/DAS

Bruno Benetti.
Avaliador
Instituição Seazone

Prof. Eduardo Camponogara, Dr.
Presidente da Banca
UFSC/CTC/DAS

Dedico este trabalho aos meus amigos e à minha família,
pelo amor e carinho incondicionais ao longo destes anos.

AGRADECIMENTOS

A realização deste trabalho não teria sido possível sem o apoio, incentivo e contribuição de diversas pessoas e instituições, às quais sou profundamente grato.

Primeiramente, gostaria de expressar minha gratidão à minha família, que sempre acreditou em mim e me apoiou incondicionalmente ao longo de toda a jornada acadêmica. Agradeço ao meu pai, Felipe Burigo, à minha mãe, Rosi Burigo, e à minha irmã, Hanna Burigo, por serem exemplos de dedicação e amor, e por me ensinarem importantes valores que carrego comigo.

Sou imensamente grato à Universidade Federal de Santa Catarina (UFSC) e a todos os professores, bem como à equipe técnica e de secretariado do Departamento de Automação e Sistemas (DAS). Suas orientações, ensinamentos e suporte foram fundamentais para o meu crescimento acadêmico e pessoal. Em especial, agradeço ao meu orientador professor Ricardo Rabelo, cuja presença e disponibilidade me guiaram tanto nos momentos de alegria quanto nos de desafio.

Aos amigos que fiz durante essa caminhada, meu profundo agradecimento por todo o carinho, apoio e incentivo. Vocês foram verdadeiros pilares e me deram a força necessária para enfrentar os desafios do curso. Em particular, destaco meus colegas da graduação em Engenharia de Controle e Automação, essa família que compartilhou comigo momentos inesquecíveis e me incentivou a jamais desistir, culminando no sonho de realizar o Linguição da Automação e a formatura.

Quero agradecer também à equipe da Seazone, empresa onde trabalho, que desde o início me acolheu com carinho e sempre buscou o meu crescimento profissional. Em especial, meu agradecimento ao Bill (Bruno Benetti) e ao André Crescenzo, cujo apoio e incentivo me impulsionaram a buscar sempre o melhor de mim.

À equipe do Pipe, composta por André Padilha, Augusto Hideki, Artur Brito, Márcio Fazolin, Lucas Abel e Nícolas Campana, e a todos que fizeram parte deste time, meu agradecimento especial. Em particular, agradeço ao Campana, cujo papel como orientador técnico foi imprescindível para a conclusão deste projeto.

Por fim, agradeço a todas as outras pessoas que, de alguma forma, contribuíram para a realização deste trabalho, seja com ideias, sugestões ou apoio moral.

A todos vocês, meu mais sincero obrigado. Vocês foram peças fundamentais nessa jornada, e sou grato por ter contado com a presença de cada um em minha vida.

Quando você faz uma descoberta – mesmo se
você for a última pessoa na Terra a ver a luz
– você jamais vai se esquecer.
(Carl Segan)

DECLARAÇÃO DE PUBLICIDADE

Florianópolis, 30 de Junho de 2023.

Na condição de representante da Seazone Serviços LTDA na qual o presente trabalho foi realizado, declaro não haver ressalvas quanto ao aspecto de sigilo ou propriedade intelectual sobre as informações contidas neste documento, que impeçam a sua publicação por parte da Universidade Federal de Santa Catarina (UFSC) para acesso pelo público em geral, incluindo a sua disponibilização *online* no Repositório Institucional da Biblioteca Universitária da UFSC. Além disso, declaro ciência de que o autor, na condição de estudante da UFSC, é obrigado a depositar este documento, por se tratar de um Trabalho de Conclusão de Curso, no referido Repositório Institucional, em atendimento à Resolução Normativa n° 126/2019/CUn.

Por estar de acordo com esses termos, subscrevo-me abaixo.

Documento assinado digitalmente
BRUNO EDUARDO BENETTI
Data: 17/07/2023 15:00:32-0300
CPF: ***.335.709-**
Verifique as assinaturas em https://v.ufsc.br

Bruno Benetti
Seazone Serviços LTDA

RESUMO

Este estudo apresenta a implementação de um processo de Integração Contínua e Entrega Contínua (CI/CD) para serviços serverless em nuvem, com foco nos serviços AWS Lambda e AWS Glue, utilizando a ferramenta Git Actions. O objetivo principal é automatizar a implantação eficiente e escalável de serviços serverless para processamento de dados em um ambiente de Data Lake. A infraestrutura necessária será provisionada por meio do AWS CloudFormatio e AWS SAM, garantindo a padronização e a replicabilidade da implantação. O fluxo de CI/CD estabelecido permitirá a integração contínua do código-fonte, execução de testes automatizados e implantação dos serviços serverless. A utilização do Git Actions como parte central do processo de CI/CD possibilitará a execução de tarefas automatizadas, como compilação, testes e empacotamento dos serviços, proporcionando maior agilidade e eficiência no ciclo de desenvolvimento.

Palavras-chave: Serveless, Data Lake, Integração Contínua, Entrega Contínua, Infraestrutura como Código

ABSTRACT

This study presents the implementation of a Continuous Integration and Continuous Delivery (CI/CD) process for serverless services in the cloud, focusing on AWS Lambda and AWS Glue services, using the Git Actions tool. The main objective is to automate the efficient and scalable deployment of serverless services for data processing in a Data Lake environment. The necessary infrastructure will be provisioned through AWS CloudFormation and AWS SAM, ensuring standardization and replicability of the deployment. The established CI/CD flow will enable continuous integration of source code, automated testing, and deployment of serverless services. The use of Git Actions as a central part of the CI/CD process will allow the execution of automated tasks such as compilation, testing, and packaging of services, providing greater agility and efficiency in the development cycle.

Keywords: Serveless, Data Lake, Continuous Integration, Continuous Delivery, Infrastructure as Code

LISTA DE FIGURAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Inicialmente, apresentar-se-á a problemática global justificadora do projeto. Ainda, neste capítulo tratar-se-á a importância da cultura de DevOps para o desenvolvivelmente do projeto infra apresentado. Finalmente, apresentar-se-ão os objetivos gerais e específicos que permeiam o projeto bem como a estrutura desenvolvida no documento.

1.1 PROBLEMÁTICA GLOBAL

De acordo com o *14º Annual State of Agile Report* (AGILE, s.d.), 95% das organizações entrevistadas utilizam pelo menos uma equipe que adota a metodologia ágil, sendo que a maioria dessas equipes está envolvida no desenvolvimento de software. No entanto, o relatório também destaca que as organizações ágeis estão adotando lentamente a cultura do DevOps.

O DevOps é uma abordagem colaborativa que busca integrar as equipes de desenvolvimento de software (Dev) e operações de TI (Ops) para melhorar a colaboração, eficiência e qualidade no ciclo de vida de desenvolvimento de software. Embora 69% dos entrevistados considerem a transformação para DevOps importante ou muito importante para suas organizações, apenas 55% delas empregam integração contínua (CI) e 41% utilizam entrega contínua (CD).

A integração contínua (CI) é uma prática que envolve a integração frequente do código-fonte de diferentes desenvolvedores em um repositório compartilhado. Por meio da CI, as alterações de código são integradas automaticamente e testadas em um ambiente automatizado, permitindo a identificação precoce de problemas e a redução de conflitos entre códigos. No entanto, apesar de sua importância, apenas uma parcela das organizações pesquisadas implementou essa prática.

Já a entrega contínua (CD) é uma extensão da integração contínua que visa automatizar o processo de entrega de software. Com o CD, as alterações de código que passaram pela integração contínua são automaticamente empacotadas, testadas e preparadas para implantação em um ambiente de produção ou de teste. A entrega contínua permite a distribuição rápida e regular de atualizações de software, mantendo a qualidade e a estabilidade do produto.

Embora as organizações reconheçam a importância do processo de ponta a ponta do DevOps, a implementação completa dessas práticas pode ser desafiadora. Cada etapa de automação requer investimento e trabalho adicional para garantir a robustez. No entanto, a tendência geral é que as empresas compreendam cada vez mais a necessidade de adotar a cultura do DevOps para melhorar a eficiência e a qualidade de seus processos de desenvolvimento de software.

1.2 OBJETIVOS

O projeto tem por foco principal o desenvolvimento inicial da implementação um sistema de de entrega contínua e integração contínua (CI/CD), como ponto de partida da efetivação da cultura de DevOps para o desenvolvimento de serviços serverless no processamento de dados em um data lake.

Importante ressaltar que para implementar o sistema de entrega contínua e integração contínua precisamos criar ambientes, infraestrutura - como códigos, entre outros. Ao final deste projeto, espera-se alcançar os seguintes objetivos específicos para criar um sistema robusto:

- Criação de um repositório do tipo Git para armazenamento e versionamento dos arquivos.
- Verificação da possibilidade de testes unitários para os serviços serveless com foco em processamento de dados.
- Criação de do pipeline automatizado de CI/CD.
- Estudo da infraestrutura em produção bem como o modo de replicação.
- Melhores práticas dentro da AWS para criação de um ambiente de desenvolvimento.

Com fulcro nesses objetivos, o projeto pretende implementar um pipeline de CI/CD, que permitirá aos desenvolvedores trabalharem em um repositório do tipo Git. Além disso, intenta concretizar uma infraestrutura replicável. Tal modelo permitirá que antes de um código ir para produção passe por um ambiente similar com a realização de testes. Após a aprovação e verificação, ocorrerá a implementação em produção.

1.3 ESTRUTURA DO DOCUMENTO

O trabalho foi estruturado da seguinte maneira:

- O Capítulo 2 apresenta demonstra o ambiente atual encontrado dentro da empresa Seazone. Ainda, a motivação para realização deste projeto.
- O Capítulo 3 explica os conceitos técnicos-teóricos e as principais tecnologias e suas finalidades dentro do projeto.
- O Capítulo 4 define os requisitos funcionais, não funcionais e os requisitos de negócios. Além disso, explana os passos para solucionar o problema proposto.
- O Capítulo 5 contém a integralidade da implementação da solução proposta anteriormente com a demonstração das principais dificuldades e problemas.

- O Capítulo 6 clarifica os resultados do projeto através de uma análise sucinta e objetiva dos dados obtidos.
- O Capítulo 7 traduz a conclusão final do projeto com a apresentação de uma síntese pessoal.

2 EMPRESA SEAZONE

O contexto deste estudo se dá na Seazone, uma empresa de gestão de imóveis em aluguel por temporada. Utiliza-se, nessa circunstância, a tecnologia para buscar a maior rentabilidade para seus clientes. Mais especificamente, aqui busca-se responder:

"Como a Seazone funciona?"

"Qual a arquitetura existente e os problemas enfrentados?"

"Por que implementar um pipeline de CI/CD?"

Neste capítulo busca-se responder as três perguntas supra.

2.1 SEAZONE

A Seazone é uma empresa fundada em 2018 por alunos egressos da Engenharia de Controle e Automação da Universidade Federal de Santa Catarina. Na época da fundação o foco era somente a gestão de imóveis de aluguel por temporada em plataformas digitais com a utilização da tecnologia para obter melhores resultados e rentabilidade para seus clientes.

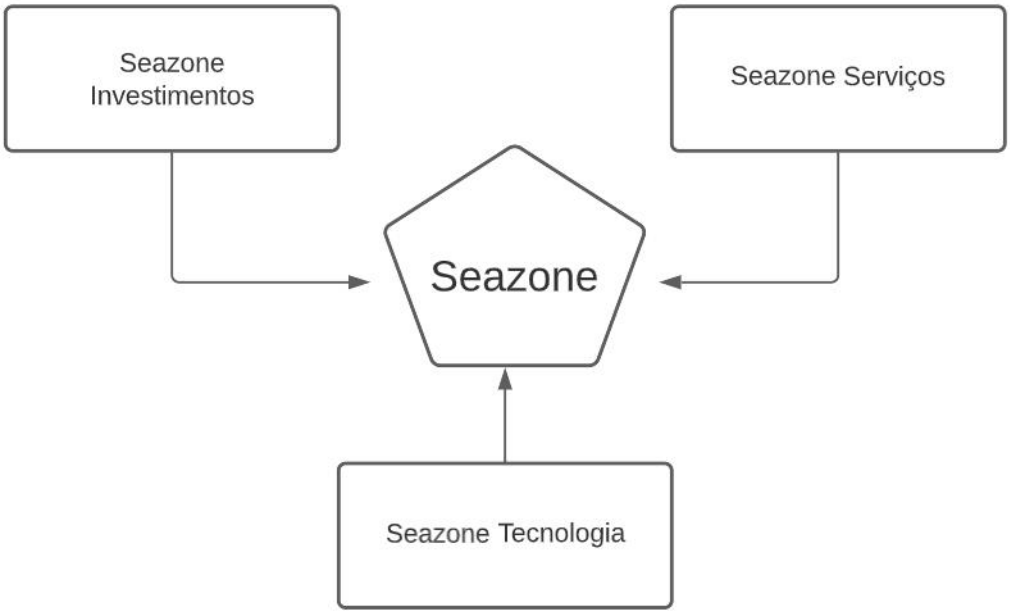
Figura 1 – Logo Seazone

Fonte: Arquivo interno Seazone

Atualmente a Seazone é uma Holding, isto é, uma entidade que detém o controle acionário de outras empresas, coordenando e administrando suas operações de forma centralizada. Sendo detentora de três grandes empresas como mostrado na Figura 2, a Seazone Serviços foi onde tudo começou, responsáveis pela administrações de imóveis por temporada. Já, a Seazone Investimentos surgiu para atender a uma necessidade específica do mercado imobiliário de identificar oportunidades lucrativas no setor de aluguel por temporada, analisar quais modelos de imóveis têm maior potencial de locação e facilitar a visibilidade de projetos de construção para atrair investidores e construtoras interessadas. Por último, a Seazone Tecnologia, o coração da empresa, desenvolve toda parte tecnológica para as demais áreas e serviços oferecidos pela

Seazone. É dentro dela que este projeto foi desenvolvido e seu funcionamento será explicado em maiores detalhes adiante.

Figura 2 – Seazone Holding



Fonte: Autor

2.1.1 Seazone Tecnologia

Dentro da Seazone Tecnologia desenvolve-se alguns produtos que vieram como necessidade das outras partes da empresa. Um dos primeiros produtos foi o SAPRON (Sistema de Administração de Propriedades Online), que consiste de um ERP (Entreprise Resource Planning) do tipo PMS (Property Management System). O sistema indicado sana a necessidade de um sistema de gerenciamento de propriedade com foco nos diferentes atores do processo, buscando diferentes soluções para usabilidade e interface. Hoje conta com um interface web, com a possibilidade de o usuário acompanhar o rendimento do seu imóvel.

Outro produto desenvolvido foi o Sirius, um produto de precificação, que surgiu das dificuldades da Seazone de alterar os preços das diárias que vendia no Airbnb.

Inicialmente, buscou-se obter dados de preço e disponibilidade para auxiliar com as decisões de precificação. Com a experiência obtida, verificou-se padrões nas formas valoração. Então, passou-se a a desenvolver códigos que automatizam esses processos. Com intuito de fazer as tomadas de decisões sem que fossem suposições era necessário dados, acompanhamento do mercado e os concorrentes para sempre

buscar a maior rentabilidade. Para isso foi criado a equipe do Pipe, que conta com engenheiros e cientistas de dados.

A equipe do Pipe foi responsável pela criação de códigos para aquisição de dados, utilizando da prática de data scraping (raspagem de dados), isto é, empregar códigos para acessar sites e obter suas informações públicas. Além de obter os dados foi necessário armazenar os mesmos. Para isso foi criado um banco de dados relacional dentro da Amazon Web Services (AWS) através da utilização do RDS (Relational Database Service) com a ferramenta de acesso de dados do PostgreSQL que possui suporte oferecido pela RDS.

Após alguns anos, esta solução de banco de dados começou a ficar obsoleta e com custos elevados. Além do Sirius que necessitava de dados, outras áreas começaram a solicitar informações, como por exemplo a Seazone Investimentos. Assim, iniciou-se a aquisição de dados não somente de sites de aluguel por temporada, mas do mercado imobiliário como um todo. Desse modo, cada vez mais o banco de dados ficava mais pesado, dificultando as consultas. Como exemplo, para acessar dados antigos, podia demorar horas.

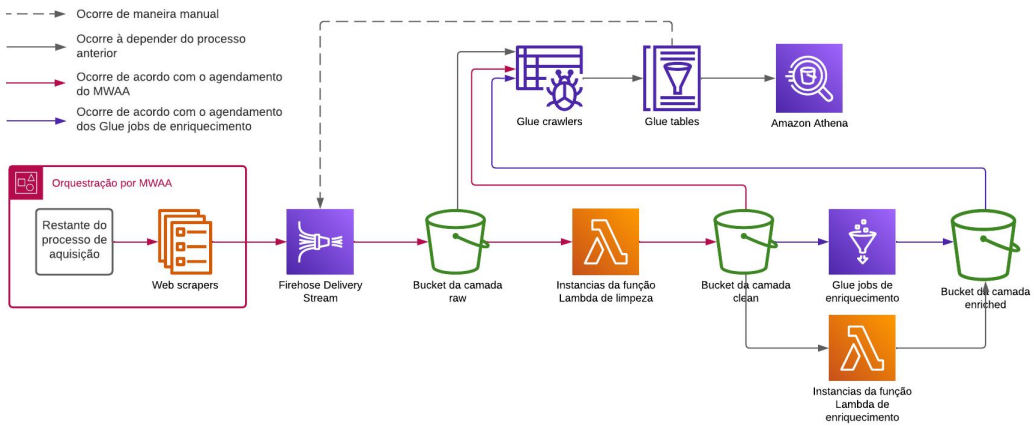
A solução encontrada foi fazer a migração para um data lake, que é um repositório centralizado que permite armazenar e processar grandes volumes de dados de forma flexível. Este proporciona recursos para análise avançada e proteção dos dados armazenados (CLOUD, s.d.).

Para a migração foi criado um conta específica na AWS, contendo somente o data lake. Isto permitiu um acesso aos dados muito mais rápido e eficiente. Com o acesso facilitado e eficiente, mais solicitações e projetos foram sendo desenvolvidos. Em suma, com a velocidade necessária, tudo foi sendo desenvolvido dentro da conta do data lake e em produção.

2.1.2 Arquitetura Existente

O data lake é divido em algumas camadas, sendo as três principais, camada raw aonde os dados chegam, não possuir tratamentos. Após essa camada são utilizados códigos para fazer a limpeza e o dado é passado para camada clean. A terceira camada é a enriched em que são utilizados códigos para realizar o tratamento e transformação dos dados da camada clean. Via de regra são dados desta última camada citada que são utilizados em análises. O fluxograma do funcionamento destas três camadas pode ser visto na Figura 3. Existem, ainda, outras camadas menores que não aparecem na Figura 3, como por exemplo o models, que possui tabelas com predições e que utilizam aprendizado de máquina para sua elaboração.

Figura 3 – Fluxograma data lake Seazone



Fonte: Arquivo interno Seazone

Dentro de cada camada, temos diversos dados divididos em tabelas, cada uma dessa tabela segue o fluxograma e possui um código para sua limpeza e se necessário seu enriquecimento. Como podemos perceber tem-se uma arquitetura baseada em microsserviços, isto é, uma abordagem organizacional de desenvolvimento de software no qual possui-se pequenos serviços independentes que se comunicam. Desse modo, facilita-se a escalabilidade e agiliza-se o desenvolvimento (AMAZON WEB SERVICES, s.d.).

Para transformar e limpar os dados são utilizados dois serviços da AWS, Funções Lambda e Glue jobs. Adianta-se que seu funcionamento pormenorizado será explicado adiante. Atualmente, a empresa conta com aproximadamente 18 códigos que rodam no AWS Lambda Functions e 17 códigos que rodam em AWS Glue Jobs. Desconsiderando as tabelas existentes dentro da camada Raw, que são dados sem tratamentos, tem-se em torno de 52 tabelas de dados limpo ou transformados pelos os serviços da AWS.

Como todo o desenvolvimento era realizado em produção, quando havia necessidade ou vontade fazer uma alteração, era necessário o fazer com o serviço em produção. Modelo que gerava erros, e quando algo não saia como o esperado, acabava quebrando outras tabelas. Era como querer modificar um carro com ele estando em movimento.

Foram desenvolvidos códigos para limpeza e enriquecimento, utilizando funções Lambda, APIs para acesso aos dados, sistema de monitoramento e outras finalidades. No entanto, isso resultou em uma situação confusa, uma vez que cada programador era responsável por escolher o estilo e formatação dos seus próprios códigos, além dos nomes utilizados, o que gerou falta de padronização. Outro problema comum era

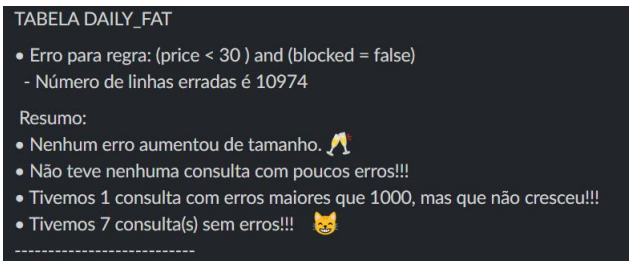
quando dois programadores alteravam o mesmo código e um acabava excluindo as modificações feitas pelo outro, o que ocasionava retrabalho.

2.1.2.1 Sistema de monitoramento de dados

Foram desenvolvidos sistemas de monitoramento de dados, buscando uma maior qualidade de dados. Os sistemas foram desenvolvidos em sua maioria utilizando as funções Lambda, eles fazem consultas que buscam varrer os dados das camadas clean e enriched. Apresentando algum valor indevido, buscamos o dado na camada que apresentou o erro, e é feita uma investigação, descendo as camadas até chegar no raw, verificando todo o caminho, importante ressaltar que a parte de investigação é feita manualmente, como a correção do erro.

As funções rodam realizando as varreduras e monitoramento e enviam relatórios diariamente no Slack (aplicativo de mensagem para empresas). Existem alguns tipos de relatórios, o primeiro tipo que pode ser visualizado na Figura 4. Neste exemplo analisa-se a tabela "Daily_fat", realizando ao total 8 consultas, mostrando se é um erro grande ou se teve algum crescimento. Todos as consultas são baseadas nos requisitos de negócios da área e buscam fazer com que a equipe seja a primeira a saber de algum erro e não o usuário final dos dados.

Figura 4 – Relatório qualidade de dados



Fonte: Arquivo interno Seazone

Um segundo exemplo de monitoramento realizado é na ingestão de dados. Aqui é analisado desde a ingestão na camada raw, passando pela clean e chegando na enriched. Com esse relatório sabe-se se algo não rodou como esperado. Nesse norte, denota-se que qualquer mudança grande no valor de ingestão de dados pode representar um erro. Logo, com esse sistema também é possível identificar quando o sistema não rodou, sendo assim retorna zero no valor de ingestão.

Pode-se verificar um exemplo na Figura 5 do relatório de ingestão. Nesse exemplo, monita-se a ingestão na camada raw, da tabela de preços e disponibilidade (priceav) do Airbnb. Do lado do número das linhas injetadas no data lake, tem-se o per-

centual de alteração nos dados, e é por essa métrica que é verificado se existe algum erro ou não.

Figura 5 – Relatório de ingestão de dados

Fonte: Arquivo interno Seazone

2.1.2.2 Sistemas de Logs

Quando um erro é apresentado, o primeiro passo é verificar em qual parte do processamento ocorreu o erro. Após a apuração do erro, é possível visualizar os logs dos serviços na AWS. Dentro da Seazone, todos os serviços na AWS estão conectados ao serviço Amazon CloudWatch. Este observa e monitora as aplicações na AWS, no qual se observa se há algum erro ou problema de execução existente.

Além disso, o serviço é utilizado para monitorar a performance da aplicação. Logo, obtém-se valores de tempo de execução, memória utilizada, entre outros fatores.

2.1.2.3 Repositório GitHub

O repositório padrão utilizado dentro da empresa é o GitHub. A equipe do pipe o utiliza em outras aplicações para além da já citada, como por exemplo no sistema de scrapers que fazem as aquisições de dados, em que todos os códigos estão dentro de um repositório no GitHub e possuem um sistema de entrega contínua.

Também já era existente um repositório para o data lake, sendo que dentro dele existia um sistema de entrega contínua para arquivos em python para Glue jobs da AWS. O sistema era bastante simples, uma pasta dentro do repositório com os arquivos em python, que podiam ser enviados ativando manualmente o pipeline do Git Actions. Assim, este empacotava os arquivos e desempacotava dentro de um bucket no S3 da conta do data lake. Deste modo, o Glue jobs conseguia identificar o arquivo em python e chama-lo para ser utilizado.

O sistema existente nunca foi colocado em produção e nenhum código foi migrado para ser utilizado. Isso pois não possuía uma infraestrutura como código, não sendo replicável e a estratégia apresentada, só sendo possível para atualizar os Glue jobs existentes. Assim, caso houvesse vontade de criar um novo era preciso entrar dentro da conta da AWS.

2.1.3 Proposta de Organização

Durante o primeiro trimestre do ano, foram realizado reuniões e levantados os principais pontos que precisáva-se executar para organizar a conta do data lake.

Um dos principais pontos era o versionamento de código, isto é, quando um código é modificado e tem-se as versões anteriores sem as modificações. Assim caso faça uma modificação que gere erros, pode-se voltar para versão anterior, fazendo assim um Rollback. Em uma tradução literal, reversão, significa descartar todos as alterações realizadas durante a transação (ALOIA; RIBEIRO, 2018).

Portanto, ao migrar os códigos para um repositório evita-se que quando dois programadores trabalham no mesmo código, ele seja implementado excluindo a alteração realizada pelo outro desenvolvedor. Cria-se, assim, uma lógica no repositório que permita a implementação de uma maneira mais automatizada e segura, traduzindo-se em um ambiente de desenvolvimento, evitando como falado anteriormente fazer alterações diretamente no código em produção. Em outras palavras, fez-se um pipeline de entrega e implementação continua e iniciou-se a implementação da cultura do DevOps.

3 FUNDAMENTAÇÃO TEÓRICA E TECNOLOGIAS UTILIZADAS

Neste capítulo, iremos nos basear no contexto apresentado no capítulo anterior para discutir os fundamentos teóricos e as tecnologias necessárias para construir uma solução elegante.

3.1 DEVOPS

DevOps, desenvolvimento e operações, é uma combinação de filosofias culturais, práticas e ferramentas que buscam aumentar a velocidade, confiabilidade e capacidade no desenvolvimento de software. Algumas dessas práticas incluem automação e simplificação dos processos, bem como o gerenciamento da infraestrutura, permitindo que as empresas inovem mais rapidamente (AWS, s.d.).

Para entender melhor a origem do termo e a necessidade das práticas de DevOps, é importante conhecer o contexto em que surgiu. Em 2003, alguns anos após a popularização da internet, o Google contratou Ben Treyor para liderar a equipe de operações em um ambiente de produção separado do ambiente de desenvolvimento. Essa equipe ficou conhecida por criar a Engenharia de Confiabilidade de Sites (SREs, Site Reliability Engineering, em inglês) (METZ, 2016) .

Embora as práticas do Google e o conceito de SREs sejam semelhantes ao DevOps, o termo e as práticas de DevOps evoluíram separadamente. Isso ocorreu porque o Google manteve essas práticas em segredo por cerca de uma década, como parte de sua estratégia corporativa (METZ, 2016).

Em 2007, Patrick Debois recebeu uma tarefa junto ao Ministério do Governo da Bélgica para auxiliar na migração de um data center. Durante esse processo, ele foi responsável por organizar as atividades e os relacionamentos entre as equipes de desenvolvimento de aplicações e a equipe de operações, que lidava com servidores e a infraestrutura de rede e bancos de dados (MEZAK, 2018).

No entanto, Debois ficou frustrado com a falta de coesão entre os métodos de desenvolvimento de aplicações e os métodos de infraestrutura. Essa frustração o levou a refletir sobre o problema e começar a buscar soluções (MEZAK, 2018).

Durante a Agile Conference em Toronto, em 2008, Andrew Schafer organizou uma reunião para discutir o tópico de Infraestrutura Ágil. No entanto, apenas Patrick Debois compareceu a essa reunião, demonstrando interesse em discutir sobre o assunto. Essa discussão inicial foi o ponto de partida para o surgimento de novas ideias e o conceito de Administração Ágil de Sistemas (MEZAK, 2018).

Pode-se afirmar que o ano de 2009 marcou a consolidação do termo DevOps, quando dois funcionários do Flickr fizeram uma apresentação intitulada *"10+ Deploys per Day: Dev and Ops Cooperation at Flickr"*. Essa apresentação foi um tanto dramática, retratando os problemas existentes entre as equipes de desenvolvimento e

operações, com acusações mútuas típicas, onde uma área culpava a outra pelos erros do sistema. A apresentação defendeu que a única maneira de resolver esses problemas era promover maior transparência e integração entre as atividades das duas áreas (MEZAK, 2018).

Atualmente, o DevOps é sustentado por cinco pilares conhecidos como CALMS (Culture, Automation, Lean IT, Measurement, Sharing). Esses pilares representam as iniciativas de cultura, automação, metodologia lean, mensuração e compartilhamento. Esses aspectos são fundamentais para a implementação efetiva do DevOps (SHARMA, 2014).

Figura 6 – CALMS Pilares DevOps

Fonte: (RESILLE, 2017)

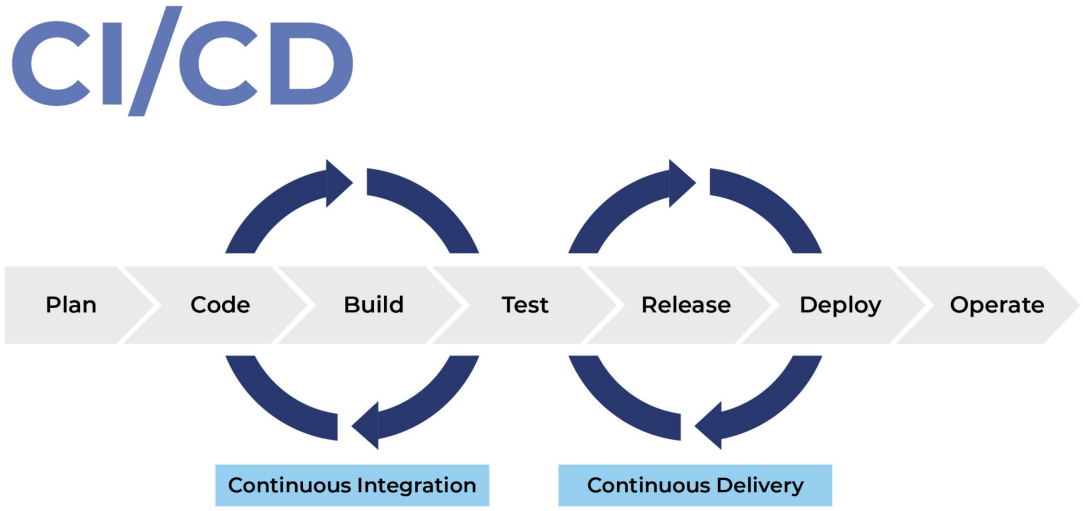
Como mencionado anteriormente nesta seção, o DevOps tem suas raízes no desenvolvimento ágil e tem como objetivo melhorar e otimizar a entrega e o desenvolvimento de software. No entanto, é importante ressaltar que o DevOps não é uma solução mágica por si só. Para obter sucesso com a implementação do DevOps, é

necessária uma mudança cultural nas organizações (SHARMA, 2014).

3.2 ENTREGA CONTÍNUA E INTEGRAÇÃO CONTÍNUA

Duas práticas essenciais dentro do fundamento do DevOps desenvolvem um papel fundamental na abordagem ágil de desenvolvimento de software: Integração Contínua e Entrega Contínua. Popularmente conhecido por CI/CD (Continuous Delivery e Continuous Integration), são parte importante para dar eficiência, segurança e qualidade ao desenvolvimento.

Figura 7 – Pipeline CI/CD



Fonte: (NAVIA, 2020)

A Integração Contínua (CI) é uma prática que permite que equipes de desenvolvimento, trabalhando em componentes de diferentes tecnologias, colaborem de forma contínua e ágil. Essa prática tem como objetivo a integração frequente e automática do código desenvolvido por diferentes membros da equipe, garantindo que qualquer alteração seja incorporada ao código-base principal de forma rápida e consistente. A CI ajuda a reduzir riscos, identificar problemas mais cedo no ciclo de vida do desenvolvimento e promover uma cultura de colaboração e responsabilidade compartilhada (SHARMA, 2014).

A Entrega Contínua (CD) é uma extensão natural da Integração Contínua que automatiza o processo de implantação do software em ambientes de desenvolvimento, testes e produção. Com a CD, as empresas podem garantir que cada alteração validada pelo processo de CI seja entregue de forma rápida e confiável em diferentes ambientes, seguindo um fluxo automatizado e consistente. A CD promove eficiência ao reduzir a

necessidade de intervenção manual na implantação do software, além de minimizar a possibilidade de erros decorrentes de processos inconsistentes (SHARMA, 2014).

Para muitos praticantes de DevOps, a ênfase está na Entrega Contínua, e é comumente vista como um elemento central das práticas DevOps. Por esse motivo, muitas ferramentas e soluções que são promovidas como ferramentas de DevOps são focadas principalmente nesse processo. No entanto, é importante destacar que o DevOps vai além da Entrega Contínua e envolve uma abordagem abrangente que inclui cultura, colaboração, automação e outras práticas para melhorar o desenvolvimento e a operação de software. Embora a Entrega Contínua seja uma parte crucial do DevOps, é importante não limitar o conceito apenas a esse aspecto e considerar as demais práticas e princípios que compõem essa abordagem (SHARMA, 2014).

3.3 TESTES

Dentro do estudo de caso proposto, existem dois principais objetivos de teste: qualidade de software e qualidade de dados. É importante entender a diferença entre esses dois tipos de testes e quais foram estudados no projeto.

3.3.1 Teste de software

O teste de unidade é considerado um dos processos mais comuns em testes, pois verifica individualmente os componentes mais simples do sistema, como métodos e classes. O objetivo é identificar falhas decorrentes de defeitos de lógica e garantir a corretude desses componentes (FERNANDES, 2017).

O teste de integração é uma técnica sistemática utilizada na construção da arquitetura de software. Ele verifica se as diferentes etapas ou componentes do sistema, quando integrados, funcionam de maneira correta e coesa. Podemos dizer que os testes de integração realizam a união dos testes de unidade, verificando se a integração entre essas partes ocorre de forma adequada (NOELLO, 2016).

Além disso, é possível aplicar ferramentas de Lint no projeto de testes. As ferramentas de Lint realizam uma análise estática do código, buscando possíveis bugs, verificando erros de sintaxe e garantindo a conformidade com padrões de estilo de escrita, como a padronização de nomes de variáveis e funções, entre outros (SANTOS, 2018).

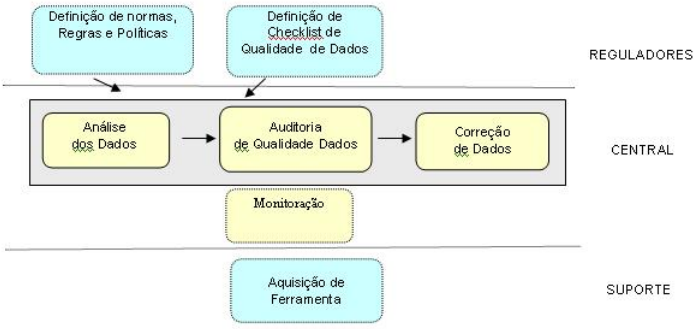
3.4 QUALIDADE DE DADOS

No processo de qualidade de dados, é necessário identificar problemas nos dados e classificá-los. Isso envolve encontrar o que chamamos de "dados sujos", que são dados que deveriam estar em uma forma limpa e consistente. Por exemplo, em uma coluna que armazena preços de imóveis, espera-se que o valor seja um número inteiro

positivo após o processo de limpeza. No entanto, se ocorrer uma falha no processo de limpeza, o valor pode ser retornado com um símbolo de cifrão e ser considerado como texto, o que o torna um dado "sujo"(CALDO, 2018).

Em uma visão geral, a qualidade de dados não se resume apenas a uma ferramenta específica. É necessário considerar como um conjunto de processos que compõem uma arquitetura de qualidade de dados. Essa arquitetura de processos de qualidade de dados pode ser visualizada na Figura 8.

Figura 8 – Arquitetura de Processos de qualidade de dados



Fonte: (CALDO, 2018)

Na análise de dados, é realizada uma varredura na base de dados para apresentar como os dados estão estruturados e organizados. Já na auditoria, verifica-se se os resultados obtidos anteriormente estão de acordo com o esperado, permitindo assim a validação dos dados (CALDO, 2018).

Conforme mencionado no capítulo anterior, na Seazone, há um monitoramento de dados implementado para realizar a análise e auditoria dos mesmos. Esse monitoramento tem como objetivo evitar a propagação de erros em todo o pipeline de dados, garantindo a qualidade e a integridade dos dados ao longo do processo.

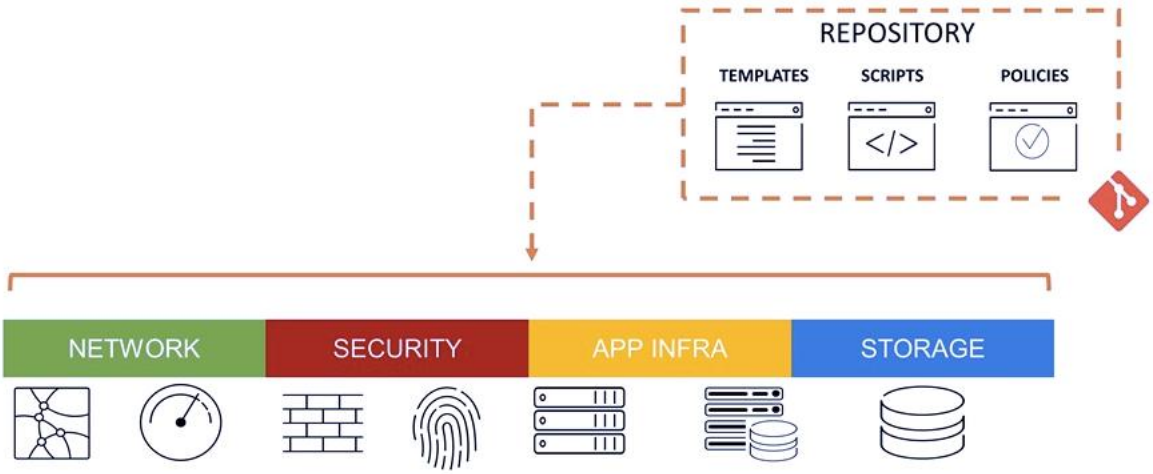
3.5 INFRAESTRUTURA COMO CÓDIGO (IAC)

A Infraestrutura como Código (IaC) é uma abordagem comum em DevOps, na qual as equipes podem gerenciar as configurações e automatizar o provisionamento da infraestrutura através de códigos escritos. Essa abordagem permite implementar e fornecer serviços de forma consistente e repetível, eliminando a necessidade de processos manuais para configurar e gerenciar a infraestrutura do sistema. Com a IaC, é possível tratar a infraestrutura como se fosse código, aplicando práticas de controle de versão, testes e automação, proporcionando maior agilidade, confiabilidade e escalabilidade ao ambiente de desenvolvimento e operação (HUMBLE; FARLEY, 2013).

Sem a infraestrutura como código, enfrentamos dificuldades para criar um CI/CD, pois não conseguimos provisionar a infraestrutura e realizar a entrega para cada ambiente específico. Caso seja necessário criar um ambiente, precisamos fazer todo o processo manualmente, e a cada alteração é necessário modificar todos os ambientes antes de implementar uma nova versão (SHARMA, 2014).

A Figura 9 ilustra a estrutura de uma IaC, na qual vários sistemas que utilizam scripts (arquivos contendo código) são executados. Esses sistemas têm suas infraestruturas provisionadas por meio de templates e arquivos que contêm políticas de autorização. A estrutura desenvolvida no projeto é bastante similar a essa representada na figura.

Figura 9 – Fluxo infraestrutura como código



Fonte: (LIMA, 2018)

3.6 GIT E GITHUB

O Git é um sistema de controle de versão distribuído amplamente utilizado na indústria de desenvolvimento de software. Ele permite que equipes de desenvolvedores rastreiem e gerenciem as alterações feitas em seu código-fonte ao longo do tempo. O Git é projetado para lidar com projetos de qualquer tamanho, desde pequenos projetos pessoais até grandes projetos empresariais.

Uma das principais características do Git é o seu modelo de ramificação e mesclagem (branching and merging), que permite que os desenvolvedores trabalhem em paralelo em diferentes recursos ou problemas sem interferir no trabalho uns dos outros.

Cada desenvolvedor pode criar sua própria ramificação (branch) para trabalhar em uma alteração específica e, em seguida, mesclar suas alterações de volta à ramificação principal (branch principal) quando estiverem prontas. Isso permite um fluxo de trabalho colaborativo e facilita a organização das alterações feitas por cada desenvolvedor, mantendo a integridade do código-fonte (ATLASSIAN, s.d.).

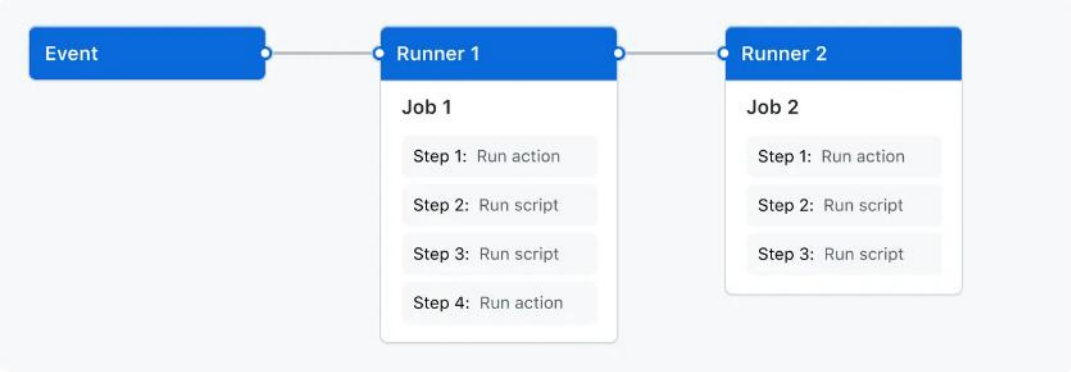
O GitHub é uma plataforma de hospedagem baseada em nuvem para repositórios Git. Ele oferece um ambiente colaborativo onde os desenvolvedores podem compartilhar, colaborar e contribuir para projetos de software. Além de ser um local para armazenar e compartilhar código-fonte, o GitHub também oferece recursos adicionais, como controle de acesso, rastreamento de problemas, solicitações de pull, revisões de código e muito mais. Esses recursos adicionais facilitam a colaboração entre os desenvolvedores, o gerenciamento de problemas e o processo de revisão de código, tornando o GitHub uma plataforma amplamente utilizada na indústria de desenvolvimento de software (GITHUB, s.d.[b]).

Os repositórios hospedados no GitHub podem ser públicos ou privados, dependendo das necessidades do projeto. Isso permite que desenvolvedores e equipes colaborem de forma eficiente em um ambiente centralizado, acompanhem alterações, gerenciem problemas e revisem o trabalho uns dos outros. O GitHub é uma plataforma amplamente utilizada pela comunidade de desenvolvedores e é frequentemente usada como uma plataforma para projetos de código aberto, bem como para projetos comerciais e empresariais (GITHUB, s.d.[b]).

3.6.1 GitHub Actions

O GitHub Actions é um serviço de automação baseado em nuvem fornecido pelo GitHub. Ele permite que os desenvolvedores automatizem fluxos de trabalho, incluindo integração contínua (CI) e entrega contínua (CD), diretamente em seus repositórios do GitHub. O GitHub Actions é uma ferramenta poderosa para automatizar tarefas repetitivas e aprimorar a eficiência do processo de desenvolvimento de software. Com o GitHub Actions, os desenvolvedores podem definir e configurar fluxos de trabalho personalizados, que são acionados por eventos específicos, como push de código, criação de pull requests ou lançamentos de novas versões. Essa automação permite a execução de testes, criação de builds, implantação em ambientes de teste ou produção, entre outras tarefas, de forma rápida e confiável. A Figura 10 ilustra a ideia geral do GitHub Actions (GITHUB, s.d.[a]).

Figura 10 – Fluxo de trabalho GitActions



Fonte: (GITHUB, s.d.[a])

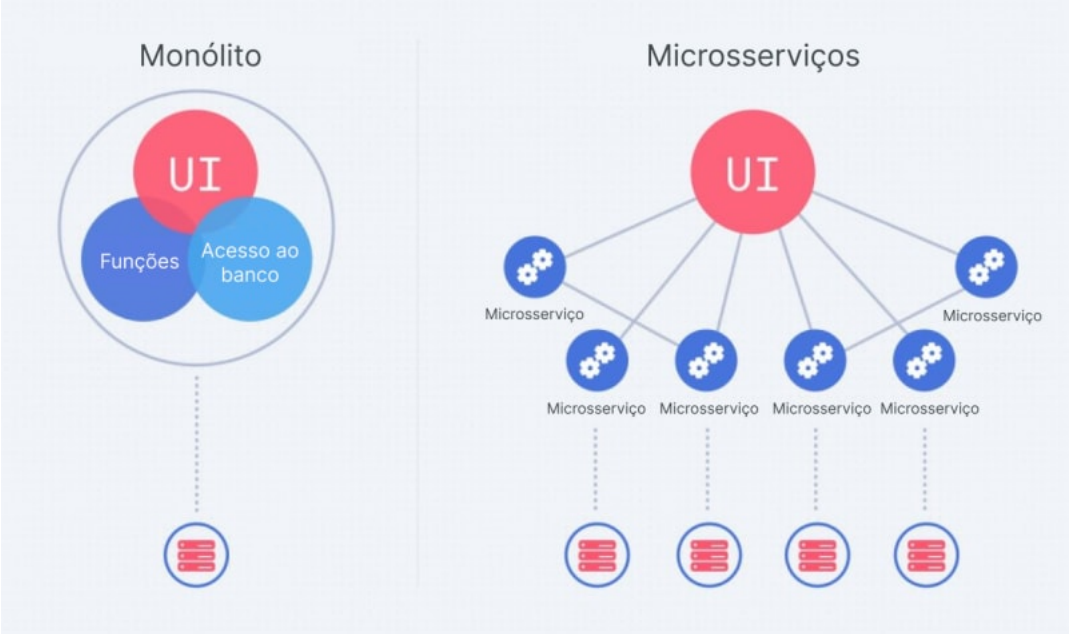
A utilidade do GitHub Actions para CI/CD reside no fato de que ele permite criar fluxos de trabalho personalizados que são executados automaticamente sempre que ocorrem eventos específicos em um repositório do GitHub. Esses fluxos de trabalho podem ser configurados com etapas (steps) que executam várias ações, como compilação de código, execução de testes automatizados, implantação em ambientes de teste ou produção e notificações. Além disso, o GitHub Actions oferece recursos avançados, como paralelização de tarefas e integração com serviços de nuvem populares, o que torna a automação de CI/CD ainda mais flexível e escalável. Com o GitHub Actions, os desenvolvedores podem automatizar o processo de construção, teste e implantação de seu software de forma confiável e eficiente (GITHUB, s.d.[a]).

3.7 MICROSSERVIÇOS

Microsserviços são uma abordagem arquitetural para desenvolver e implantar aplicações, em que o sistema é dividido em componentes independentes e autônomos, chamados de microsserviços. Cada microsserviço é responsável por uma funcionalidade específica do aplicativo e pode ser desenvolvido, implantado e escalado de forma independente dos outros microsserviços. Essa abordagem permite uma maior flexibilidade, modularidade e escalabilidade, além de promover a reutilização de código e facilitar a manutenção do sistema (AMAZON WEB SERVICES, s.d.).

Em contraste, uma aplicação monolítica é construída como uma única unidade de software, em que todas as funcionalidades são desenvolvidas e implantadas juntas. No modelo monolítico, o aplicativo é um único código-base, normalmente escrito em uma única linguagem de programação, e todas as partes do sistema compartilham o mesmo ambiente de execução. Isso pode tornar o desenvolvimento e a manutenção do aplicativo mais complexos, além de dificultar a escalabilidade, uma vez que qualquer alteração no código requer a implantação de todo o aplicativo (KANCZUK, 2020).

Figura 11 – Diferença microsserviço e monolítico



Fonte: (BUENO, 2022)

A diferença fundamental entre microsserviços e sistemas monolíticos está na sua estrutura e granularidade. Enquanto os microsserviços são componentes independentes e autônomos, que se comunicam entre si por meio de interfaces bem definidas, os aplicativos monolíticos são uma única entidade, onde todas as partes estão intimamente acopladas. Essa diferença afeta aspectos como desenvolvimento, implantação, escalabilidade e manutenção dos sistemas, tornando os microsserviços mais flexíveis, modulares e adequados para ambientes complexos e escaláveis, enquanto os aplicativos monolíticos podem ser mais simples de serem desenvolvidos e implantados, mas podem apresentar desafios na medida em que o sistema cresce e evolui (KANCZUK, 2020).

3.8 SERVERLESS

Serverless, traduzindo significa "sem servidor". Isso se refere a aplicações em que a lógica do lado do servidor é escrita e desenvolvida pelo desenvolvedor da aplicação, sem a necessidade de se preocupar diretamente com a infraestrutura do servidor. Essas aplicações são normalmente ativadas por eventos e podem ser totalmente gerenciadas por terceiros. Nesse modelo, os desenvolvedores podem se concentrar na implementação da lógica de negócio da aplicação, enquanto a infraestrutura de servidor é gerenciada automaticamente pelo provedor de serviços de nuvem. Isso proporciona maior agilidade, escalabilidade e redução de custos, já que os recursos são

alocados de acordo com a demanda, sem a necessidade de provisionar e gerenciar servidores de forma manual (ROBERTS, 2018).

Os benefícios de redução de custos no modelo Serverless podem ser percebidos pelos usuários em dois aspectos principais. O primeiro aspecto é a economia de custos na infraestrutura. Ao adotar um ambiente Serverless, os usuários podem se beneficiar da capacidade de aproveitar uma infraestrutura compartilhada, o que reduz os custos de aquisição e manutenção de servidores dedicados. Isso significa que os usuários não precisam investir em hardware ou lidar com tarefas de gerenciamento de servidores, resultando em economias significativas (ROBERTS, 2018).

O segundo aspecto está relacionado aos ganhos de eficiência e redução de custos trabalhistas. Ao utilizar um sistema Serverless fornecido por um provedor de serviços, os usuários podem economizar tempo e recursos que seriam normalmente gastos no desenvolvimento e hospedagem de um sistema equivalente por conta própria. Isso ocorre porque os provedores de serviços gerenciam toda a infraestrutura e garantem a escalabilidade automática, o que elimina a necessidade de provisionar e configurar manualmente servidores. Além disso, os provedores de serviços também são responsáveis por tarefas de monitoramento, segurança e manutenção da infraestrutura, o que reduz a carga de trabalho dos usuários e permite que eles se concentrem exclusivamente no desenvolvimento de suas aplicações (ROBERTS, 2018).

Função como Serviço (Function as a Service - FaaS) é um modelo de computação em nuvem que faz parte do paradigma Serverless, onde os desenvolvedores podem escrever e implantar pequenas funções de código que são executadas em um ambiente totalmente gerenciado pelo provedor na nuvem. Nesse modelo, os desenvolvedores se concentram apenas na lógica do código, sem a necessidade de se preocuparem com a infraestrutura subjacente, como provisionamento de servidores ou dimensionamento. A FaaS oferece uma abordagem altamente escalável e elástica, permitindo que as funções sejam acionadas automaticamente em resposta a eventos específicos (ROBERTS, 2018).

3.9 DATA LAKE

Um Data Lake é um repositório centralizado que armazena grandes volumes de dados brutos, estruturados e não estruturados, em sua forma original. Diferentemente dos sistemas tradicionais de armazenamento de dados, que exigem uma estruturação prévia dos dados antes de serem armazenados, um Data Lake permite a ingestão de dados em seu estado bruto, preservando sua integridade e flexibilidade (CLOUD, s.d.).

No Data Lake, os dados são geralmente armazenados em formatos semiestruturados, como JSON, CSV ou Parquet, e podem ser provenientes de diversas fontes, como bancos de dados, dispositivos IoT, mídias sociais, logs de servidores, entre outros. Essa abordagem de armazenamento de dados permite que as organizações

capturem e retenham um amplo conjunto de informações sem a necessidade imediata de definir uma estrutura rígida, facilitando a exploração e análise posterior desses dados para diferentes propósitos, como análise de negócios, aprendizado de máquina e tomada de decisões estratégicas (CLOUD, s.d.).

3.10 LINGUAGEM DE MARCAÇÃO YAML

YAML (acrônimo de "YAML Ain't Markup Language") e YML (extensão de arquivo comumente associada ao YAML) são formatos de serialização de dados que permitem representar informações de maneira legível tanto para humanos quanto para máquinas. YAML é uma linguagem de marcação simples e intuitiva, usada para estruturar e organizar dados em uma estrutura hierárquica (IMPACTA, 2021).

O GitHub Actions, serviço de integração contínua e entrega contínua do GitHub, utiliza YAML para definir e configurar os workflows. Um workflow é um conjunto de ações automatizadas que ocorrem em resposta a eventos específicos, como push de código para um repositório.

No GitHub Actions, o arquivo YAML é denominado "arquivo de configuração do workflow"e é usado para descrever as etapas a serem executadas, os gatilhos de eventos, as condições, as configurações de ambiente e outras opções relevantes para a execução do fluxo de trabalho.

Por sua vez, os templates do AWS CloudFormation também podem ser escritos em YAML ou JSON. Esses templates são arquivos que descrevem a infraestrutura e os recursos da AWS que precisam ser criados ou gerenciados pelo CloudFormation.

Os templates do CloudFormation permitem especificar detalhes como instâncias EC2, grupos de segurança, funções do Lambda, tabelas do DynamoDB e muito mais. Ao executar o CloudFormation, ele lê o template e provisiona automaticamente os recursos descritos nele.

3.11 AMAZON WEB SERVICES

A AWS (Amazon Web Services) é uma plataforma de serviços em nuvem oferecida pela Amazon.com. Ela foi lançada oficialmente em 2006 e se tornou uma das maiores e mais populares plataformas de computação em nuvem do mundo.

A história da AWS começa no início dos anos 2000, quando a Amazon.com estava buscando uma solução escalável para lidar com o crescente tráfego em seu próprio site. Isso levou a empresa a perceber que poderia usar sua infraestrutura de TI existente para fornecer serviços em nuvem a outras empresas, aproveitando sua escalabilidade e capacidade de gerenciamento (MARQUES, 2023).

A oferta inicial da AWS consistia em serviços básicos, como armazenamento e computação em nuvem, chamados de Amazon S3 (Simple Storage Service) e Amazon

EC2 (Elastic Compute Cloud), respectivamente. Esses serviços foram pioneiros na ideia de infraestrutura como serviço (IaaS), em que os clientes podem alugar recursos de computação em vez de comprá-los e mantê-los fisicamente (MARQUES, 2023).

Com o tempo, a AWS expandiu seus serviços para incluir uma ampla variedade de opções, como bancos de dados, análise de dados, inteligência artificial, aprendizado de máquina, internet das coisas, segurança e muito mais. A empresa também investiu em infraestrutura global, construindo data centers em várias regiões do mundo para fornecer serviços de nuvem com baixa latência e alta disponibilidade (MARQUES, 2023).

A AWS ganhou popularidade entre startups, empresas de médio porte e grandes corporações devido à sua flexibilidade, escalabilidade e modelo de pagamento por uso. Ela permite que as empresas dimensionem rapidamente seus recursos de TI conforme necessário, sem a necessidade de investir em hardware e infraestrutura física.

Vamos utilizar esta seção para apresentar as principais aplicações dos serviços utilizados pela Seazone para o processamento de dados do Data Lake e para a realização deste projeto. A seguir, serão destacados os principais casos de uso desses serviços.

3.11.1 Amazon S3

O Amazon S3 (Simple Storage Service) é um serviço de armazenamento em nuvem oferecido pela AWS (Amazon Web Services). Ele foi lançado em 2006 como um dos primeiros serviços da AWS e se tornou uma opção amplamente utilizada para armazenamento de dados na nuvem (MARQUES, 2023).

O Amazon S3 é projetado para fornecer armazenamento altamente escalável, durável e seguro para uma ampla variedade de aplicativos. Ele permite que os usuários armazenem e recuperem grandes quantidades de dados de forma eficiente, seja para backup, arquivamento, distribuição de conteúdo ou para alimentar aplicativos e serviços em nuvem (AMAZON WEB SERVICES, 2023b).

A principal finalidade dentro da Seazone é o armazenamento de dados. O data lake está armazenado dentro dele, em buckets (baldes). Sendo por eles que dividimos nosso data lake em camadas, cada bucket representando uma delas. Além disso, armazenamos arquivos para as mais diversas aplicações (AMAZON WEB SERVICES, 2023b).

3.11.2 Aws Glue

AWS Glue é um serviço de preparação e transformação de dados oferecido pela Amazon Web Services (AWS). Ele permite aos usuários descobrir, catalogar e transformar dados de maneira eficiente para análise. Com recursos automatizados de descoberta de metadados e geração de código, o Glue simplifica o processo de ETL

(Extração, Transformação e Carga) em um data lake. Com sua interface intuitiva, o Glue permite aos usuários definir fluxos de trabalho personalizados para extrair, transformar e carregar dados em diferentes formatos, tornando-se uma ferramenta valiosa para processamento de dados em larga escala (AMAZON WEB SERVICES, 2023d).

Dentro do Aws Glue podemos utilizar o Apache Spark, um framework de processamento de dados em larga escala e de alto desempenho. Ele oferece uma abstração de programação flexível e extensível para processar grandes volumes de dados distribuídos em um cluster de computadores. Sua arquitetura distribuída e seu mecanismo de processamento em memória garantem um processamento eficiente e rápido de dados (AMAZON WEB SERVICES, 2023d).

Unindo AWS Glue e o Apache Spark, conseguimos criar fluxos de trabalho de processamento e transformação de dados em um data lake. O AWS Glue atua como um catálogo de metadados e uma ferramenta de geração de código para os processos de ETL, enquanto o Spark é responsável pela execução das transformações em larga escala. Essa combinação permite a descoberta automática de metadados, a execução eficiente de transformações em grandes volumes de dados (AMAZON WEB SERVICES, 2023d).

3.11.3 AWS Lambda Functions

AWS Lambda é um serviço de computação em nuvem oferecido pela Amazon Web Services (AWS) que permite executar código sem a necessidade de provisionar ou gerenciar servidores. Ele segue o modelo de "Function as a Service"(Função como Serviço), onde os desenvolvedores podem carregar seu código e o Lambda cuidará de todo o dimensionamento automático e infraestrutura subjacente. Isso permite que os desenvolvedores se concentrem exclusivamente na lógica de negócios de suas funções, sem se preocuparem com o gerenciamento de servidores ou com a escalabilidade da aplicação (AMAZON WEB SERVICES, 2023e).

Ao usar o AWS Lambda, os desenvolvedores podem escrever suas funções em várias linguagens de programação populares, como Python, JavaScript, Java, C e outras. Essas funções são acionadas por eventos, como alterações em um bucket do Amazon S3, atualizações em uma tabela do Amazon DynamoDB ou eventos personalizados. Quando um evento ocorre, o Lambda provisiona dinamicamente os recursos necessários para executar a função e, em seguida, desativa esses recursos quando a execução é concluída. Isso garante que os desenvolvedores paguem apenas pelo tempo de execução real de suas funções, sem nenhum custo adicional pela infraestrutura subjacente (AMAZON WEB SERVICES, 2023a).

O AWS Lambda é utilizado para criar arquiteturas de microsserviços, desenvolver aplicativos sem servidor e automatizar tarefas de processamento de dados. Ele oferece flexibilidade e escalabilidade, permitindo que as funções sejam executadas

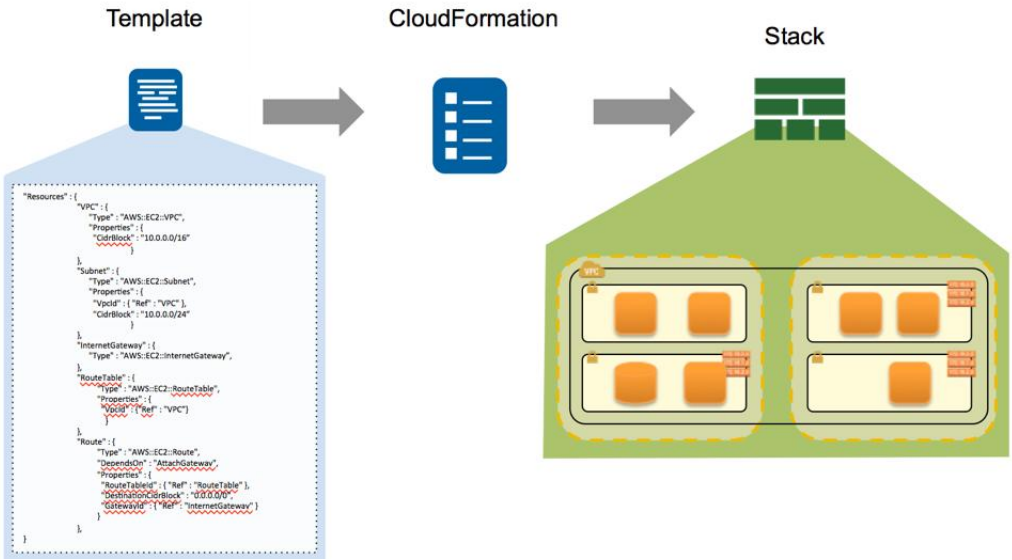
de maneira eficiente em resposta a uma ampla variedade de eventos. Além disso, o Lambda se integra perfeitamente com outros serviços da AWS, como o Amazon API Gateway, o Amazon S3 e o Amazon DynamoDB, permitindo que os desenvolvedores criem aplicativos altamente eficientes e robustos na nuvem.

3.11.4 Aws CloudFormation

AWS CloudFormation é um serviço oferecido pela Amazon Web Services (AWS) que permite provisionar e gerenciar recursos de infraestrutura de forma automatizada e consistente. Ele segue o conceito de "infraestrutura como código", onde a infraestrutura é definida e gerenciada por meio de arquivos de texto chamados templates. Os templates do CloudFormation são escritos em formato JSON ou YAML e descrevem os recursos necessários, como Aws Lambda Functions, buckets no AWS S3, grupos de segurança, entre outros (AMAZON WEB SERVICES, 2023c).

Ao usar o AWS CloudFormation, é possível criar e gerenciar conjuntos de recursos relacionados chamados de stacks. Um stack é uma unidade lógica de provisionamento, que representa um conjunto de recursos que devem ser criados, atualizados ou excluídos juntos. O CloudFormation garante que os recursos definidos no template sejam criados e configurados corretamente, mantendo o estado desejado da infraestrutura ao longo do tempo (AMAZON WEB SERVICES, 2023c).

Figura 12 – Estrutura Cloud Formation



Fonte: (DOROSZ, 2019)

Ao criar uma stack, o CloudFormation realiza automaticamente as ações necessárias para criar e configurar os recursos especificados no template. Além disso, ele

permite a atualização das stacks existentes para refletir alterações no template, aplicando as mudanças de forma segura e com baixo impacto. Essa abordagem simplifica a gestão da infraestrutura e reduz o risco de erros humanos durante a implantação (AMAZON WEB SERVICES, 2023i).

Nested stacks (ou "stacks aninhados") são uma funcionalidade avançada do AWS CloudFormation que permite criar hierarquias de stacks, onde um stack pode ser incorporado como um componente dentro de outra stack principal (AMAZON WEB SERVICES, 2023h).

A stack principal é responsável por orquestrar os componentes aninhados, referenciando os templates correspondentes. Quando a stack principal é criada, ela cria e gerencia automaticamente as stacks aninhados, passando os parâmetros necessários e tratando as dependências entre elas. Isso permite a divisão da infraestrutura em partes menores e gerencie as interações entre elas de forma mais eficiente (AMAZON WEB SERVICES, 2023h).

3.11.5 AWS SAM

O AWS SAM (Serverless Application Model) é uma extensão do AWS CloudFormation, projetada especificamente para simplificar o desenvolvimento e implantação de aplicativos sem servidor na AWS. Ele oferece uma sintaxe simplificada e mais expressiva para descrever recursos de aplicativos sem servidor, como funções Lambda, APIs do API Gateway e tabelas do DynamoDB. O SAM permite definir a infraestrutura como código e oferece recursos adicionais para testar, depurar e gerenciar aplicativos sem servidor (AMAZON WEB SERVICES, 2023g).

Uma das principais vantagens do AWS SAM é a facilidade de desenvolvimento e teste local. Ele fornece um ambiente de teste local que permite que os desenvolvedores executem e depurem suas funções Lambda em seus próprios computadores antes de implantá-las na nuvem. Isso acelera o ciclo de desenvolvimento e melhora a eficiência ao reduzir a necessidade de implantações completas durante o desenvolvimento e teste (AMAZON WEB SERVICES, 2023g).

Além disso, o AWS SAM oferece integração com ferramentas populares de CI/CD (Continuous Integration/Continuous Deployment) e fluxos de trabalho de desenvolvimento, como o AWS CodePipeline e o AWS CodeDeploy. Isso facilita a integração do SAM em pipelines de entrega contínua, permitindo que os aplicativos sem servidor sejam implantados de forma automatizada e consistente em diferentes ambientes (AMAZON WEB SERVICES, 2023g).

Na realização do projeto, será utilizado a sintaxe do AWS SAM e sua estrutura. Mas vale ressaltar que quando enviamos os templates para AWS, estamos enviando para Cloud Formation, ele que faz a compilação e cria a infraestrutura dentro da conta da AWS.

3.11.6 AWS IAM

IAM, sigla para Identity and Access Management, é um serviço fornecido pela Amazon Web Services (AWS) que permite gerenciar e controlar o acesso aos recursos da AWS de forma segura. O IAM desempenha um papel fundamental na segurança e governança dos ambientes de nuvem, pois permite definir políticas de acesso granulares, criando e gerenciando identidades (como usuários, grupos e funções) e atribuindo permissões específicas a essas identidades (AMAZON WEB SERVICES, 2023f).

Com o IAM, é possível controlar quem pode acessar quais recursos da AWS e quais ações podem ser realizadas nesses recursos. Ele fornece uma camada adicional de proteção, permitindo a autenticação de usuários e a aplicação de políticas de autorização baseadas em necessidades específicas.

4 REQUISITOS DO PROJETO E SOLUÇÃO PROPOSTA

Sabendo que estamos construindo um CI/CD para serviços que fazem processamento de dados para um data lake, vamos apresentar neste capítulo os requisitos definidos no início do projeto e que se esperava cumprir ao final do projeto.

Apresentaremos a proposta de solução adotada para a implementação do processo de CI/CD na Seazone, levando em consideração os requisitos e restrições identificados. Serão apresentadas as alternativas estudadas e a motivação por trás da escolha final.

Além disso, ao final do capítulo, serão incluídos diagramas que visam proporcionar um melhor entendimento ao leitor.

4.1 REQUISITOS

A definição desses requisitos veio em grande parte das reuniões com o time de desenvolvedores da Seazone, como comentado no capítulo 2, a criação de um CI/CD era algo desejado por todos na equipe.

Requisitos funcionais são as capacidades e funcionalidade específicas que o sistema, produto ou serviço deve oferecer. Considerando as boas práticas de DevOps apresentada anteriormente, conseguimos listar 4 principais requisitos funcionas para serem realizados neste projeto, eles podem ser visto no Quadro 1.

Quadro 1 – Requisitos funcionais.

Integração com o repositório Git	O CI/CD deve ser capaz de se integrar a um repositório Git, como por exemplo o GitHub, para obter o código-fonte do serviço serverless.
Disparo automático de pipelines	O CI/CD deve ser capaz de iniciar automaticamente pipelines de implantação sempre que houver uma alteração no repositório Git.
Implantação de serviços serverless	O CI/CD deve ser capaz de implantar automaticamente os serviços serverless na nuvem, utilizando ferramentas como o AWS CloudFormation ou o AWS Serverless Application Model (SAM).
Configuração de ambientes	O CI/CD deve permitir a configuração de diferentes ambientes (por exemplo, desenvolvimento, teste, produção) para implantar os serviços serverless em cada ambiente separadamente.

Fonte: Autor.

Requisitos não funcionais são critérios que definem as características e qualidade do sistema. Para este projeto, conseguimos levantar quatro requisitos principais, que podem ser vistos no Quadro 2.

Quadro 2 – Requisitos não funcionais.

Desempenho	O CI/CD deve ter uma boa velocidade de execução, garantindo tempos de resposta rápidos durante o processo de implantação.
Segurança	O CI/CD deve garantir a segurança dos dados e do ambiente de implantação, seguindo as melhores práticas de segurança recomendadas pela AWS.
Monitoramento	O CI/CD deve fornecer informações e métricas sobre o processo de implantação, permitindo o monitoramento contínuo do estado do pipeline.
Escalabilidade	O CI/CD deve ser capaz de lidar com um grande volume de implantações e ser escalável de acordo com as necessidades do sistema.

Fonte: Autor.

Foram definidos alguns requisitos de negócios, que são os requisitos que derivam das necessidades e objetivos do negócio. Refletindo as restrições, metas e prioridades. Foram levantadas dois principais requisitos que podem ser visto no Quadro 3

Quadro 3 – Requisitos de negócios.

Conformidade com padrões e políticas	O CI/CD deve seguir as políticas e padrões definidos pela empresa, garantindo a conformidade durante o processo de implantação.
Automatização	O CI/CD deve automatizar ao máximo as tarefas de implantação, minimizando a intervenção manual e reduzindo a chance de erros.

Fonte: Autor.

Como estamos implementando um CI/CD para o data lake, sem ter praticado previamente as práticas comuns de DevOps, muitos dos requisitos levantados foram baseados na filosofia de DevOps, o que os torna um pouco generalistas.

Além disso, outros requisitos de negócios foram identificados, como a medição do número de retrabalhos e bugs encontrados na execução dos códigos implementados pelo CI/CD. Essa medição visa avaliar se a implementação do CI/CD foi capaz de melhorar a qualidade dos códigos. Também é importante considerar a redução dos custos da AWS, porém, para uma análise precisa, é necessário somar os custos de todos os ambientes (Desenvolvimento, Testes, Produção) juntamente com a implementação do CI/CD e as práticas de DevOps.

Entanto, esses dois requisitos, assim como outros, não foram mencionados no Quadro 3, pois seria necessário um tempo significativo para realizar uma análise completa. Para obter resultados significativos, seria necessário um período mínimo de um semestre. Portanto, qualquer afirmação relacionada a esses requisitos seria apenas uma suposição.

4.2 SOLUÇÃO PROPOSTA

Importante salientar que a metodologia de desenvolvimento aplicada dentro da Seazone é o SCRUM, uma metodologia de desenvolvimento ágil. Essa abordagem orienta a definição do back-log do projeto com base nos requisitos funcionais, que se tornam 'histórias' a serem implementadas. No entanto, é fundamental ressaltar que são os requisitos não funcionais que ditam 'como' e 'onde' essas 'histórias' serão implementadas.(SERGIO, 2014)

Dentro desta metodologia, temos as sprint, que são, dentro da Seazone, um período de duas semanas, sendo atribuídas algumas tarefas do back-log para serem realizadas. Após o termino da sprint, é apresentado o que foi feito e, então, verificadas as criticas do resultado. Seguindo esta metodologia, muitas das soluções inicias foram alteradas no desenvolvimento do projeto. Portanto, podemos dizer que esta é a solução de proposta final.

4.2.1 Repositório e Pipeline de CI/CD

Dentro da Seazone o repositório Git utilizado é o GitHub. Sendo assim a principal escolha para construção do pipeline de CI/CD foi o GitHub Actions, ferramenta do GitHub para construção de fluxos de trabalho. Essa integração nativa entre o GitHub e o GitHub Actions proporciona uma transição mais suave e facilita a configuração e a adoção do processo de CI/CD.

Dentro da AWS, temos ferramentas para construção de fluxos de trabalho para CI/CD, como por exemplo AWS CodeBuild e o AWS CodePipeline. Estas poderiam fornecer uma solução abrangente, por estar dentro da AWS, como utilizar o mesma camada raw para os ambientes de desenvolvimento e produção, separando as camadas apenas após o primeiro processamento de dados.

No entanto, esta solução possui um alto custo de AWS, pois teria praticamente os dados replicados, não sendo no momento uma solução aplicável pelo custo. Além disso, tornaria a empresa mais dependentes da AWS, dificultando uma possível migração para outro serviço de cloud.

Além disso, o AWS CodePipeline possui valores de custos mensais por pipeline construído, além de custos pelo uso. Enquanto o GitHub Action, possui somente valores de custos por uso.

4.2.2 Infraestrutura como código e integração com AWS

Após uma análise criteriosa, opta-se, portanto, por utilizar o AWS SAM (Serverless Application Model) em conjunto com o AWS CloudFormation para operar a infraestrutura como código e integrar adequadamente com os serviços da AWS.

Uma das principais razões para a escolha do AWS SAM e não somente o AWS CloudFormation para construir um sistema de alta qualidade é que ao utilizar o AWS SAM, consegue-se descrever a infraestrutura necessária para a aplicação de forma declarativa, tratando-a como código - o que oferece uma série de benefícios - bem como através do uso do AWS CloudFormation, pode-se criar, gerenciar e provisionar a infraestrutura de forma automatizada, consistente e escalável.

É importante ressaltar que, ao implementar a infraestrutura como código, é necessário abordar a infraestrutura como um todo, e não apenas os componentes individuais, como lambdas e AWS Glue. Ao considerar a infraestrutura como um todo, garante-se que as conexões e dependências entre os serviços permaneçam intactas durante as atualizações e evita-se problemas de desconexão de eventos. Dessa forma, pode-se manter a integridade e a funcionalidade do sistema em todas as alterações.

Ao adotar essa abordagem, a equipe de desenvolvimento terá mais controle e visibilidade sobre a infraestrutura, facilitando a manutenção, o monitoramento e o rastreamento de alterações. Além disso, o uso do AWS SAM e do AWS CloudFormation simplifica a implantação e a integração contínua, permitindo que as atualizações da infraestrutura sejam realizadas de forma ágil, segura e replicadas para todos os ambientes existentes.

4.2.3 Estudo de viabilidade de testes para serviços de processamento de dados

Para tomar decisões e considerar a possibilidade de aplicar testes, é importante entender como os códigos são escritos para o processamento de dados. No caso da arquitetura de microsserviços, cada código possui uma finalidade específica, focada no processamento de dados para uma determinada tabela.

Diferentemente de outros sistemas, nossos códigos são escritos como um bloco único, geralmente contendo apenas uma função ou classe. Dentro desse bloco, há um comando para leitura dos dados necessários, utilização de bibliotecas para o tratamento e, por fim, um comando para enviar os dados para o local de armazenamento apropriado.

Considerando essa estrutura, podemos concluir que a aplicação de testes de unidade ou testes de integração neste momento é desnecessária, pois a maioria dos códigos funciona como uma unidade independente.

No entanto, como os códigos não passavam por verificações e cada programador tinha liberdade para usar nomenclaturas e padrões de sua preferência, o uso de

ferramentas de linter pode ser bastante útil. No caso da infraestrutura como código, utilizamos o AWS SAM, que possui uma verificação de linter incorporada. Isso significa que se algum recurso estiver com erro ou fora do padrão, ele não será empacotado e enviado para a AWS.

Quanto aos códigos dos serviços, como mencionado, eles estão em um estado desorganizado. Aplicar uma ferramenta de linter desde o início resultaria na reprovação da maioria dos códigos. Portanto, decidimos que no pipeline de CI/CD será incluída uma etapa de testes, mas não implementaremos os testes em si até que os códigos sejam refatorados e padronizados.

Percebemos que, para um sistema de processamento de dados, os testes mais importantes são os de qualidade de dados. Na produção, já temos monitoramento e análise dos dados. Ao aplicar os testes na conta de desenvolvimento, é importante verificar se as regras de negócio permanecem as mesmas, dependendo se os dados nessa conta serão uma réplica exata dos dados de produção ou apenas uma amostra.

No momento inicial, foi decidido que os dados não serão migrados, a menos que haja uma demanda específica. Portanto, os testes para analisar a qualidade dos dados devem ser realizados manualmente pelos desenvolvedores antes de implementar qualquer alteração na produção.

4.2.4 Organização da conta na AWS

No processo de definição da infraestrutura na AWS, uma consideração importante foi a da organização da conta AWS. Esta inicialmente tinha a intenção de usar uma única conta e separar os ambientes de desenvolvimento e produção por meio do uso do AWS SAM e do AWS CloudFormation. Essa abordagem inicial visava facilitar a gestão e o controle dos recursos, mantendo-os dentro do mesmo contexto.

No entanto, à medida que o tempo passava, percebeu-se que essa solução poderia se tornar complicada e resultar em uma conta com muitos recursos, tornando-a potencialmente desorganizada e de difícil gerenciamento. Além disso, a separação entre ambientes de desenvolvimento e produção usando somente recursos de segregação poderia levar a conflitos e problemas durante as atualizações e implantações.

Diante dessas considerações, decidiu-se adotar uma abordagem diferente: a criação de duas contas distintas na AWS. A conta atual seria designada como a conta de produção, enquanto uma réplica seria criada para servir como o ambiente de desenvolvimento. Essa separação em contas distintas oferece uma clara demarcação entre os ambientes, permitindo maior controle, governança e isolamento.

É importante ressaltar que, ao replicar os recursos na conta de desenvolvimento, não se replicou os dados. Essa abordagem visa evitar custos excessivos e manter a integridade dos dados. A migração dos dados será realizada apenas quando necessário, sob demanda, garantindo que os dados estejam atualizados e em sincronia com

o ambiente de produção. A replicação de recursos gera poucos custos de AWS, visto que a grande maioria dos serviços utilizados são cobrados por uso.

Com essa nova estrutura de contas, ganha-se maior flexibilidade para realizar testes, experimentos e implementações de novos recursos no ambiente de desenvolvimento, sem comprometer a estabilidade e a disponibilidade do ambiente de produção. Além disso, essa separação clara entre contas facilita o controle de custos, monitoramento e governança dos recursos em cada ambiente, contribuindo para uma infraestrutura mais organizada e gerenciável.

4.2.5 Estratégia de permissões GitHub-AWS

Para conseguir se conectar às contas da AWS é necessário permissão da mesma. No caso de conectar o GitHub à conta da AWS existem algumas maneiras de como fazer.

A primeira é a que era realizada na conta do data lake no GitHub antes do início deste projeto. Nessa solução, foi criado um usuário dentro da conta da AWS, utilizando o AWS IAM. Essa conta possuía um usuário e senha que eram adicionados ao GitHub, o que permitia que ele se conectasse. Nesse caso, se tivesse um vazamento desses valores, um usuário poderia usufruir disso para acessar a conta via terminal de comando.

A segunda solução, e que foi implementada, utiliza a criação de uma "role", isto é, uma identidade que possui determinada permissões atribuídas a ela. E para conseguir conectar ela ao GitHub utilizar dentro do AWS IAM usa-se o provedor de identidade (idP). Com o idP autoriza-se o GitHub a utilizar a role e suas permissões. A grande vantagem em comparação com a outra opção, é que nas políticas de acesso da "role"consegue-se configurar qual repositório e branch podem acessá-la. Assim evita-se que caso alguém consiga acesso ao valor dela possa utilizar sua máquina para invadir a conta da AWS.

4.2.6 Representação Visual da Solução Proposta

Na seção "Representação Visual da Solução Proposta", apresentaremos diagramas que ajudarão a ilustrar o funcionamento do sistema proposto. O primeiro diagrama abordado é o diagrama de caso de uso, como mostrado na Figura 13.

O diagrama de caso de uso representa o fluxo de interações entre os atores e o sistema. Nele, podemos observar as etapas do fluxo de trabalho do GitHub Actions para automatizar o processo de CI/CD.

No início, o Desenvolvedor realiza um "push"para o repositório, acionando o fluxo de trabalho. Em seguida, o ambiente de execução é iniciado automaticamente pelo GitHub Actions, permitindo que o código seja testado e analisado, inclusive com

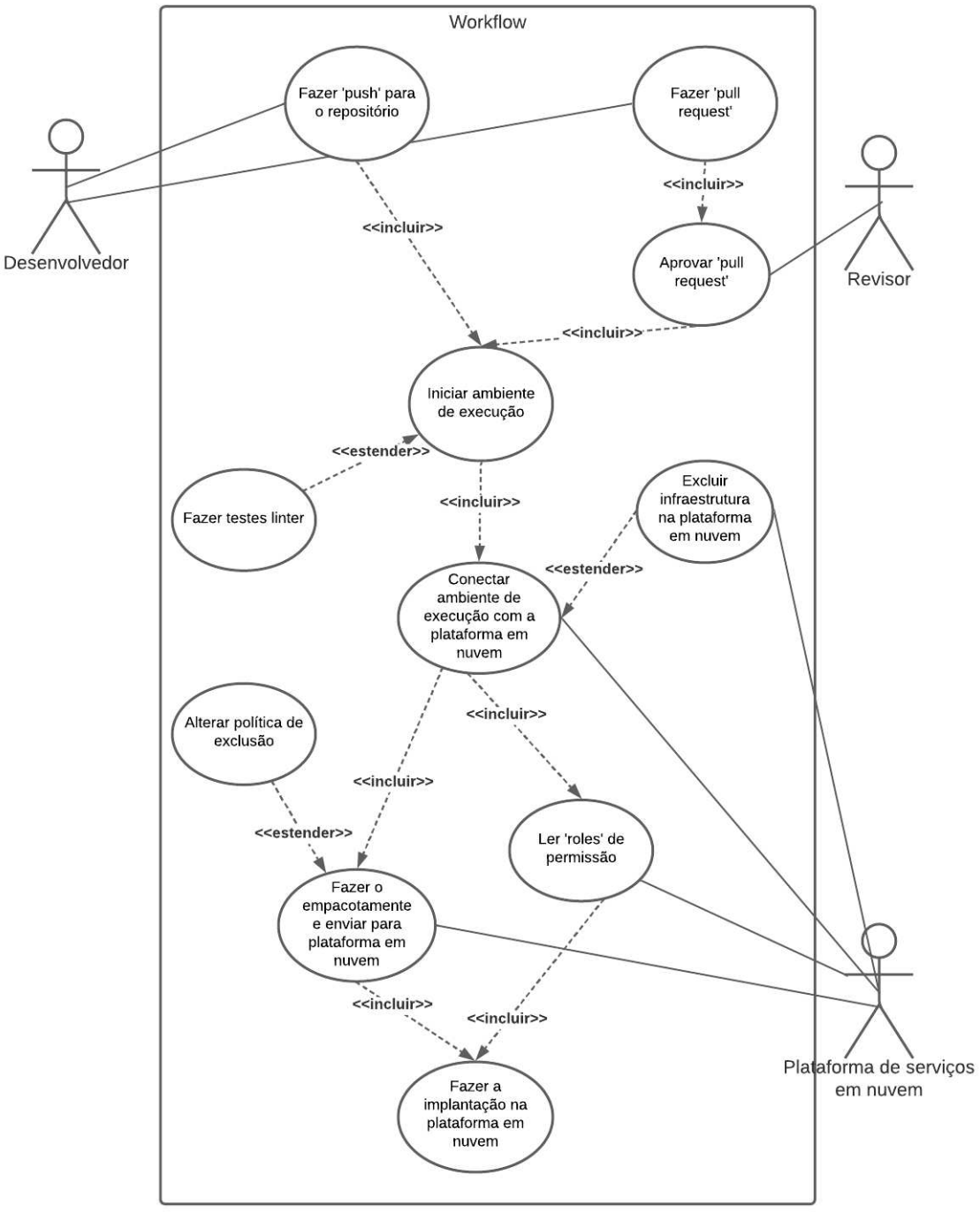
a execução de testes linter para garantir a conformidade do código com as diretrizes estabelecidas.

O Revisor desempenha o papel de revisar o código e aprovar o "pull request"criado pelo Desenvolvedor. Após a aprovação, o fluxo de trabalho continua sua execução, realizando tarefas adicionais, como o empacotamento do código e a implantação na plataforma de serviços em nuvem.

Por fim, a plataforma de serviços em nuvem é responsável por hospedar e executar o código. O GitHub Actions se integra com essa plataforma, garantindo a implantação automatizada do código no ambiente apropriado.

Assim, o diagrama de caso de uso proporciona uma visão geral do fluxo de trabalho do GitHub Actions e como ele automatiza as etapas de CI/CD dentro do sistema proposto.

Figura 13 – Diagrama de caso de uso



Fonte: Autor.

O segundo diagrama apresentado é o diagrama de arquitetura do sistema, como ilustrado na Figura 14.

No topo do diagrama, temos os Desenvolvedores e os Revisores, que desempenham papéis fundamentais no processo de desenvolvimento e revisão de código.

Eles interagem diretamente com o GitHub Repositórios, onde o código-fonte do projeto é armazenado. No repositório, são criadas e gerenciadas as Branches, sendo que a Branch feature é sempre criada como um clone da Branch dev. Essa abordagem permite que os desenvolvedores trabalhem em recursos específicos de forma isolada.

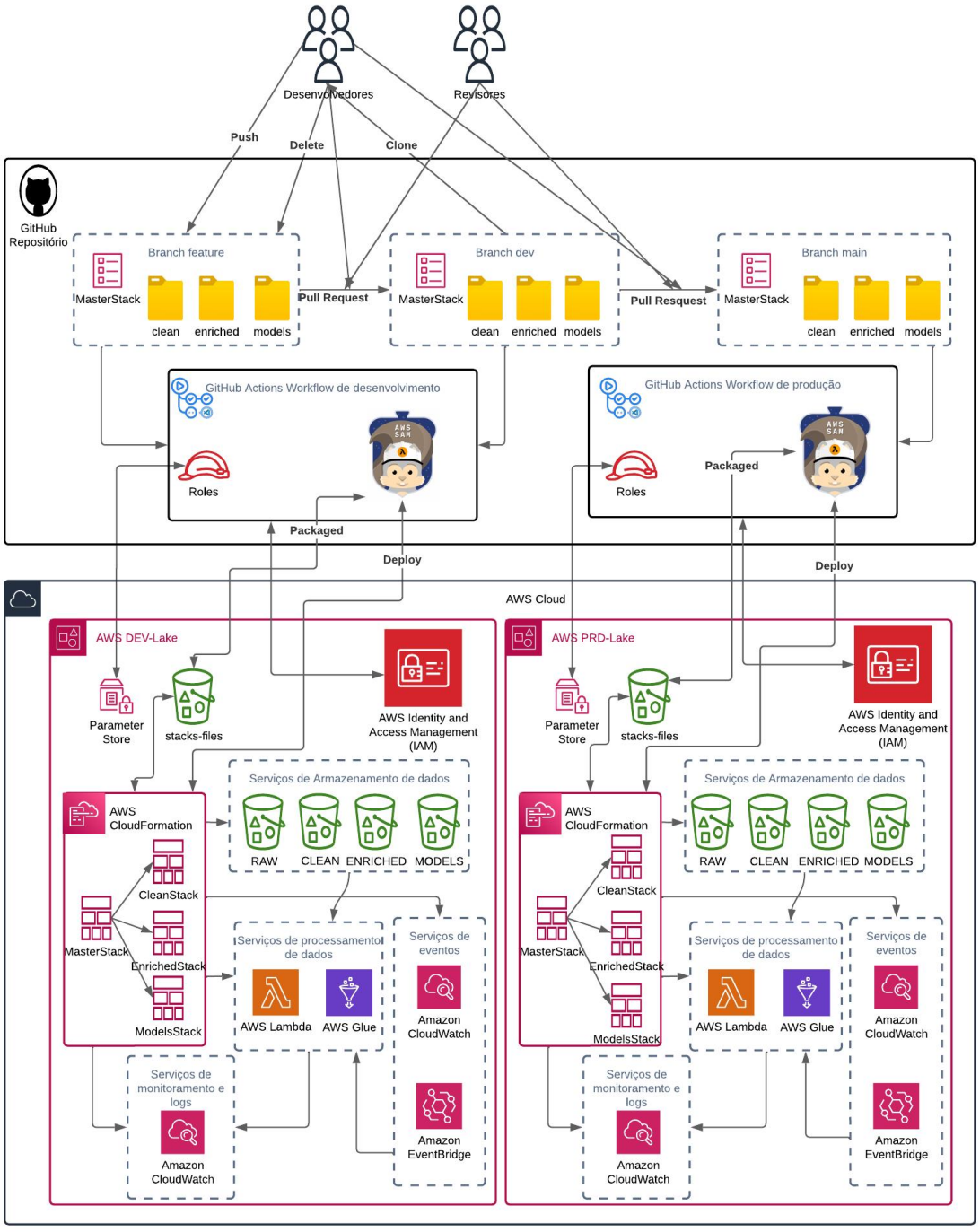
Quando um Desenvolvedor faz um "push"para a Branch feature, o código é enviado para o GitHub Repositórios, desencadeando a ativação do fluxo de trabalho. Nesse momento, é iniciada a infraestrutura na AWS DEV-lake, que é a conta de desenvolvimento. Essa infraestrutura é criada especificamente para a Branch feature e permite que os desenvolvedores realizem testes e validações do código.

Após o desenvolvimento na Branch feature ser concluído, é feito um "pull request"para a Branch dev. Isso inicia o processo de revisão do código pelo Revisor. Após a revisão e a aprovação do "pull request", o código é mesclado na Branch dev e a infraestrutura associada à Branch feature é excluída na AWS DEV-lake.

Enquanto isso, a infraestrutura da Branch dev é mantida na AWS DEV-lake, pois essa Branch representa uma versão estável do código em desenvolvimento. Essa infraestrutura serve como um ambiente de teste para a integração de diferentes recursos e para garantir a compatibilidade e o funcionamento adequado do código em conjunto.

Quando a Branch dev está pronta para ser implantada em produção, é feito outro "pull request"para a Branch main. Esse processo também envolve a revisão e a aprovação do código pelo Revisor. Após a aprovação, o código é mesclado na Branch main e, em seguida, é iniciada a implantação na AWS PRD-lake, que é a conta de produção. Nessa etapa, a infraestrutura correspondente à Branch main é criada na AWS PRD-lake para suportar a execução do código em produção.

Figura 14 – Diagrama de arquitetura



Fonte: Autor.

5 IMPLEMENTAÇÃO

Neste capítulo apresentar-se-á a implementação da solução apresentada anteriormente. Mostra-se, nesse advento, as principais dificuldades, problemas e soluções encontradas.

5.1 REPOSITÓRIO

Dentro da conta da Seazone no Github foi utilizado o repositório existente para o data lake, mas desconsiderando o que já havia sido realizado. Dentro dele está organizado todos os códigos para os serviços Lambda e Glue, os templates de IAC e códigos de workflows para GitHub Actions. Seguindo princípios de DevOps, ele segue algumas regras de organizações e modelos para implantação.

O primeiro passo foi criar duas branchs, uma principal com nome de main (produção) e outra com nome de dev (desenvolvimento). A branch main sempre está atualizada com o sistema em produção, isto é, os códigos presentes nesta branch são os que estão rodando em produção. Já, os códigos presentes na branch dev são os códigos que estão presentes no ambiente de desenvolvimento.

Além disso, temos uma política de criação para novas branchs, em que serão alterados os códigos para serem integrados a branch de desenvolvimento. A política escolhida foi a mais simples possível, tendo em vista que não existia nenhuma até o momento presente. Para nomear as branchs é preciso seguir o padrão de nome, colando entre umas das quatro opções apresentada no Quadroo 4.

Quadro 4 – Categoria branchs.

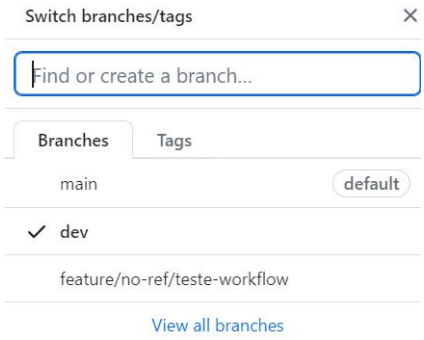
feature	Utilizada para adicionar, otimizar ou remover recursos e funcionalidades.
bugfix	Quando for realizar correções de bug.
hotfix	Alterar o código de maneira rápida, mostrando um grau de urgência, normalmente utilizada em emergências.
test	Opção para realizar testes e experimentar códigos, que estão fora de um problema ou funcionalidade adicional.

Fonte: Autor.

Ao criar uma nova Branch, o Desenvolvedor segue algumas convenções específicas para fornecer informações relevantes sobre o objetivo do ramo. Além de indicar a categoria, como "feature", é necessário adicionar uma "/"seguida pela referência à issue (pendências) correspondente, caso exista. Caso não exista uma issue associada, é adicionado "no-ref"para indicar essa ausência. Em seguida, é incluída uma descrição concisa que sintetize o propósito e objetivo da Branch, utilizando -"para separar as palavras.

Exemplifica-se: para adicionar uma funcionalidade como para ler arquivos .parque, a nova branch ficaria: feature/no-ref/adicionar-leitura-arquivos-parquet. Pode-se ver o exemplo de como o repositório está organizando pela Figura15. Temos as duas branchs principais e fixas main e dev, e uma branch para adicionar e testar os workflows. Ainda insta ressaltar que para enviar para as duas branchs fixas, é preciso da aprovação de pelo menos outro desenvolvedor, evitando assim que erros possam passar para essas branchs.

Figura 15 – Exemplo branchs GitHub



Fonte: Autor.

Além da políticas para organização das branchs, outra política importante é para organizações de commits. Commits são utilizadas para enviar para salvar as alterações dentro de uma mesma Branch. Assim, quando se faz uma commit, pode-se adicionar uma mensagem ao mesmo. Verifica-se, nesse sentido, 4 categorias que podem ser vistas no Quadro 5.

Quadro 5 – Categoria commits.

feature	Utilizada para adicionar recursos novos.
fix	Quando for realizar correções de bug.
refactor	Alterar o código com foco em desempenho ou conveniência.
chore	Para todo o restando, como escrever documentação, formatação , adicionar testes, remover códigos inúteis, entre outros.

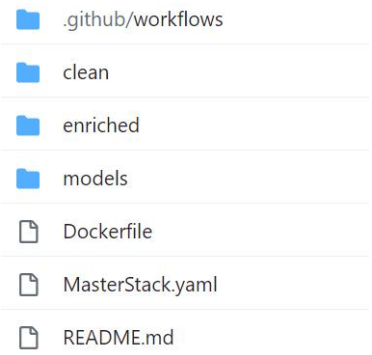
Fonte: Autor.

Diferentemente da branch, após a categoria coloca-se ":"e escreve-se a descrição do que os commit está realizando. Cada afirmação deve começar com uma declaração e caso tenha mais que uma separar por ";". Por exemplo, quando adicionar a funcionalidade de um botão ficaria: feat: adicionar novo componente de botão; adicionar botão ao template de IAC (SANOU, 2022).

Para a organização dos arquivos dentro das branchs, utiliza-se pastas. Uma para os workflows desenvolvidos, sendo que cada camada do data lake possui uma

pasta com o respectivo nome, com os códigos e o template de sua infraestrutura. O template principal que implementa os outros fica no diretório raiz, em conjunto com arquivos de documentações e outras necessidades, como pode ser visto na 16.

Figura 16 – Organização arquivos GitHub



Fonte: Autor.

5.2 WORKFLOW DE CI/CD

Como supramencionado, utiliza-se o GitHub Actions como ferramenta de automação e orquestração do processo de CI/CD. O GitHub Actions é uma solução nativa do GitHub que permite criar fluxos de trabalho automatizados diretamente no repositório do projeto.

Uma das características principais do GitHub Actions é que ele é executado em nuvem. Isso significa que não há precupação em configurar ou manter infraestrutura própria de execução. O GitHub Actions oferece um ambiente pronto para uso, fornecendo uma máquina virtual Linux para executar as etapas definidas no fluxo de trabalho.

Essa máquina virtual Linux permite ter controle total sobre as funções executadas no fluxo de trabalho. Pode-se, assim, utilizar comandos e scripts personalizados, como mencionado anteriormente, para realizar tarefas como compilação de código, execução de testes, implantação em ambientes de produção e muito mais.

Além disso, o GitHub Actions oferece uma variedade de recursos e funcionalidades que nos permite definir e personalizar nossos fluxos de trabalho de acordo com as necessidades do projeto. Assim, é possível configurar gatilhos para acionar o workflow em eventos específicos, como push de código para o repositório, criação de pull requests ou até mesmo agendamentos regulares.

Para o projeto foram criados dois fluxos separados de trabalho, um para o ambiente em desenvolvimento e outro para o ambiente em produção. Evita-se, dessa maneira, qualquer risco de implementação de um ambiente em outro. Nos dois work-

flows, o funcionamento é similar – a diferença é que o workflow de desenvolvimento possui algumas funcionalidades a mais. Por isso será efetuada uma explicação com utilização de exemplos deste workflow, citando as diferenças para o outro.

A primeira definição dentro de um workflow é como ele pode ser ativado. Dentro do GitHub podemos fazer isso no começo do arquivo como mostrado na Figura 17. É definido que ele é ativado quando é enviado ("push") para as branchs "dev"ou alguma branch que comece com "feature". O comando "paths"possui três atributos. Logo, para ser ativado a alteração feita precisa seguir algum desses formatos de arquivo.

O primeiro comando é para ser ativado somente em alterações feitas dentro de pastas e não no diretório raiz, evitando assim que caso um arquivo de documentação seja adicionado acione o workflow. Como o stack principal está no diretório raiz, há necessidade de adicioná-la nesse comando. O terceiro comando possui um "!"em seu início, isso indica proibição. Assim qualquer alteração nos arquivos do workflow não aciona o mesmo. Ainda, tem-se um comando para quando deletar uma branch que seja uma branch "feature", no qual será acionado, o motivo disso será explicado mais adiante.

Figura 17 – Ativação workflow

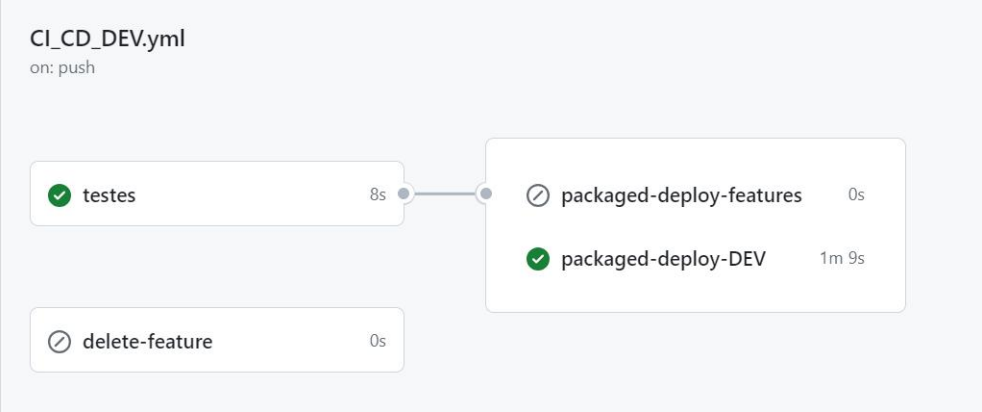
```yaml
name: Deploy AWS

on:
  push:
    branches:
      - 'dev'
      - 'feature**'
    paths:
      - '*/**'
      - 'MasterStack.yaml'
      - '!.github/workflows/**'
  delete:
    branches:
      - 'feature**'
```

Fonte: Autor.

Para o caso do workflow de produção, a única branch permitida é a "main", o comando dos "paths"são os mesmos e não possui uma ativação para branchs deletadas. Após acontecer a ativação do workflow, podemos configurar os "jobs", isto é, tarefas que são ativadas seguindo alguns padrões de especificação, como por exemplo de qual branch o código está vindo ou qual foi o evento de ativação.

Para o caso do workflow de desenvolvimento, possui-se quatro tarefas que podem ser vistas na Figura 18. Cada tarefa tem uma finalidade, a primeira chamada de testes inicia a máquina do linux e instala a biblioteca de testes do python, possibilitando aos desenvolvedores inserirem testes para os códigos nessa parte do workflow.

Figura 18 – Jobs workflow dev

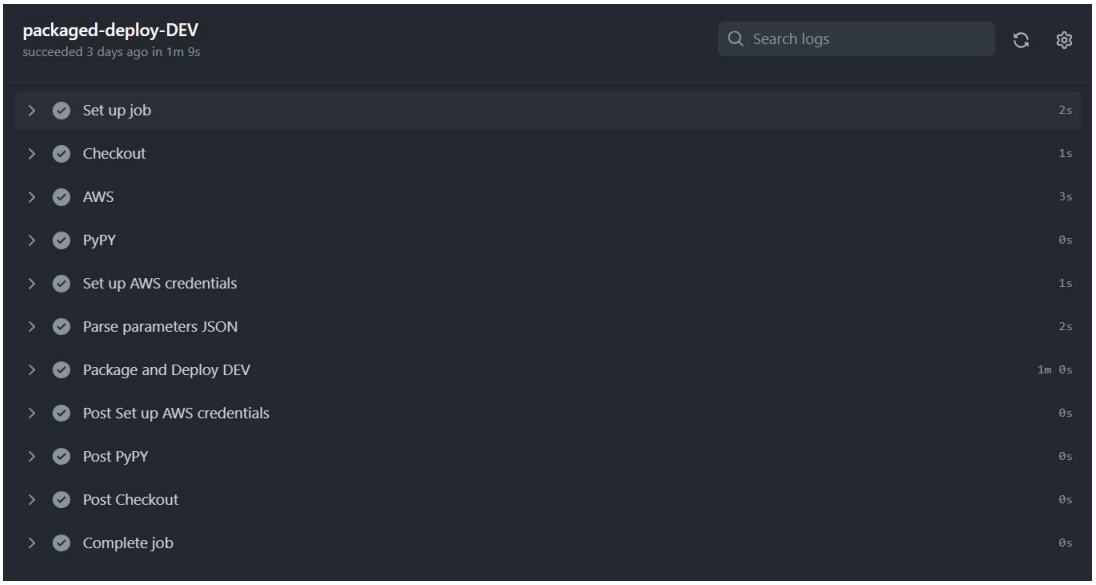


Fonte: Autor.

Sendo aprovado em testes ele pode executar alguns dos comandos de "packaged"dependendo de qual branch a ativação partiu. Dento de packged-deploy-DEV possuimos "steps"que é a divisão de ações que precisam ser executadas, sendo nos casos aplicados nesse projeto em sequência para conseguir enviar as alterações para conta DEV. Vê-se na Figura 19 todas as etapas presentes.

As quatro primeiras etapas que aparecem são para configurar o ambiente, as bibliotecas necessárias e copiar o repositório para dentro deste ambiente.) step "Set up AWS credentials". Nesse ponto, configura-se e passa-se as permissões para conectar a conta da AWS. No próximo passo ("Parse parameter JSON") são baixados os arquivos de permissão que são utilizados dentro dos recursos da IAC, evitando que nos códigos de IAC precisem conter as "roles"de permissão, e podem ser alteradas de conta para conta. O passo mais importante é "Package and Deploy DEV". Aqui que utiliza-se o AWS SAM para fazer a leitura da MasterStack e enviar para conta da AWS. Os seguintes passos são para desligar o sistema.

Figura 19 – Steps GitHub Actions



Fonte: Autor.

Todos os Job de "package"seguem esta estrutura, inclusive o do workflow de produção. Considerando os 2 pipeline de CI/CD como um todo, podemos considerar que o sistema segue alguns passos.

A criação de uma branch de "feature", ao enviar ela para o GitHub e assim criar na conta de DEV a infraestrutura para teste preliminares da "feature", sendo um espaço livre para alterações do desenvolvedor. Verificando que a "feature"está funcionando e realizando o que é esperado, o desenvolvedor pode enviar para a branch "dev"e as atualizações serão adicionadas na conta de desenvolvimento, no ambiente que consideramos ser a copia do que está em produção e que não permite alterações diretas no ambiente da AWS. Quando esta "feature"é enviada para branch "dev", a branch de feature é deletada, e toda a infraestrutura criada para "feature"na conta da aws é deletada automaticamente utilizando AWS SAM delete.

É importante ressaltar que a infraestrutura criada em dev nunca é deletada, somente a criada para os testes da "feature". Quando verificado dentro de dev que o sistema não gerou problemas é enviado para a branch de produção e assim atualizando a conta da AWS de produção.

A principal motivação para a realização disto, é que para um ambiente de data lake, os principais testes envolvem analisar a qualidade dos dados gerados, e serão implementados testes de qualidade de dados no ambiente de desenvolvimento. Assim o ambiente criado para "feature"será utilizado para criação e testes de código e o ambiente de desenvolvimento para teste de qualidade de dados. Por esse motivo o ambiente de desenvolvimento nunca é deletado e simula sempre o de produção. Neste

trabalho o ambiente de teste de qualidade de dados não foram desenvolvidos, pois as ferramentas existentes na AWS para isso são novas e extremamente caras, precisando ser feita uma análise maior e com mais tempo.

5.3 INFRAESTRUTURA COMO CÓDIGO

A infraestrutura como código é um componente essencial para a implementação do processo de CI/CD. Nesse contexto, utiliza-se o AWS CloudFormation juntamente com o AWS SAM (Serverless Application Model) para descrever e provisionar a infraestrutura de forma automatizada e consistente.

O AWS CloudFormation é um serviço da AWS que nos permite definir a infraestrutura em um formato declarativo, utilizando um template YAML ou JSON. Esses templates contêm a descrição dos recursos necessários para nossa aplicação, como lambdas, buckets do S3, tabelas do DynamoDB, Funções Lambda, Glue jobs, entre outros. Ao executar o template do CloudFormation, a infraestrutura é criada de acordo com a definição especificada, garantindo consistência e reprodutibilidade em todo o processo.

Para facilitar ainda mais a definição da infraestrutura serverless, utilizamos o AWS SAM. O AWS SAM é uma extensão do CloudFormation, projetada especificamente para desenvolvimento e implantação de aplicações serverless. Ele nos fornece uma sintaxe simplificada e recursos adicionais, como modelos de aplicativos predefinidos, que aceleram e simplificam a criação de infraestrutura serverless.

Ao estruturar os templates, adota-se uma abordagem modular, com uma pilha principal chamada MasterStack e pilhas aninhadas para as diferentes camadas do lake. Essas camadas, como clean, enriched e models, representam as etapas de processamento de dados no fluxo de trabalho.

A pilha MasterStack é responsável por orquestrar as pilhas aninhadas, garantindo a criação e configuração adequadas de cada camada. Cada pilha aninhada é projetada para lidar com uma funcionalidade específica, como a limpeza dos dados (clean), a aplicação de enriquecimentos (enriched) e a definição dos modelos de dados (models).

Uma das maiores dificuldades durante a migração da infraestrutura existente para uma infraestrutura como código, foi adequação da arquitetura existente, para uma arquitetura replicável. Como por exemplo para implementar os eventos de ativação das funções Lambda do clean. No caso, quando um arquivo do tipo .parquet chega no AWS S3 é ativado o lambda de limpeza que lê esse arquivo e faz sua limpeza enviado o .parquet para a camada clean.

Seguindo o padrão far-se-ia a configuração deste evento de ativação dentro do template do clean e conectar-se-ia ao recurso do lambda correspondente. Mas para isso recurso do bucket que armazena esses dados precisaria estar no mesmo

template, o que por questões organizacionais não é interessante, pois se dois lambdas em camadas diferentes precisarem de ligação pode gerar erros.

A solução encontrada foi definir todos os buckets dentro do template da MasterStack e fazer o próprio bucket ativar o lambda. Assim diferente do que é feito em que o evento de ativação está no recurso do lambda, dá-se permissão para o bucket ativar o lambda, adicionando assim o recurso de ativação ao bucket como pode ser visto na Figura 20.Como é visto, passamos para o bucket a função lambda que precisa ser ativada, e qual regra para sua ativação. Os valores como no exemplo advêm da stack clean.

Figura 20 – Evento de ativação lambda

```
################## Bucket RAW  #################
  BucketRaw:
    Type: AWS::S3::Bucket
    Properties:
      NotificationConfiguration:
        LambdaConfigurations:
          - Event: s3:ObjectCreated:Put
            Function: !GetAtt CleanStack.Outputs.CleanOLXArn
            Filter:
              S3Key:
                Rules:
                  - Name: prefix
                    Value: olx_listings/
                  - Name: suffix
                    Value: .parquet
```

Fonte: Autor.

Outro ponto fundamental é a organização da estrutura dentro da MasterStack. É nela que os outros templates são chamados e construídos, como pode ser visto na Figura 21. Todos os parâmetros que precisam ser passados para dentro da stack precisam ser definidos anteriormente.

Figura 21 – Estrutura Stacks

```
##################### STACKS ################################
  CleanStack:
    Type: AWS::Serverless::Application
    Properties:
      Location: ./clean/CleanStack.yaml
      Parameters:
        BucketCleanName: !Ref  BucketClean
        RoleClean: !Ref RoleClean
  EnrichedStack:
    Type: AWS::Serverless::Application
    Properties:
      Location: ./enriched/EnrichedStack.yaml
      Parameters:
        BucketRawName: !Ref BucketRaw
        BucketCleanName: !Ref  BucketClean
        BucketEnrichedName: !Ref BucketEnriched
        BucketGlueAssetsName: !Ref BucketGlueAssets
        RoleEnriched: !Ref RoleEnriched
        RoleGlue: !Ref RoleGlue
```

Fonte: Autor.

Os parâmetros são passados são os nomes do buckets, que por padrão para uma mesma organização não podem possuir o mesmo nome. Assim mesmo tendo duas contas na AWS, eles possuem nomes diferentes. Apresenta-se essa situação como uma vantagem de replicabilidade de se utilizar infraestrutura como código, dentro de uma Stack em que o recurso possui um nome lógico. Situação exemplificada, na Figura 20, em que o bucket possui o nome de BucketRaw, e em todos os ambientes dentro da stack ele vai possuir este nome, mas na conta da AWS ele recebe um nome físico, que não pode ser o mesmo para o mesmo atributo, como deixamos que AWS escolha esse nome evita erros de implementação e garante a replicabilidade do sistema.

Como comentado, o nome físico vai ser diferente dentro de cada ambiente (desenvolvimento, produção). Durante a migração essa situação se mostrou um problema, pois dentro do código chamamos o nome físico para realizar certas tarefas. A solução encontrada, foi alterar em todos códigos, para o nome físico ser passado por parâmetros ou variáveis de ambiente. Assim quando a infraestrutura é construída é configurada com o nome correto, garantindo a implementação sem erros.

Outro problema encontrado durante a implementação foi para criação e posteriormente para deletar a criação da estrutura para "features"dentro da conta de desenvolvimento. Quando define-se os buckets por padrão de configuração do AWS SAM, precisa-se passar a política de deletar. Isso precisa ser uma valor e não uma variável. Nessa toada, por padrão colocamos como "Retain", isto é, se a stack for deletada, o bucket não é. Assim, evitando que por algum erro deletar a stack de produção ou de desenvolvimento acabe deletando os dados existentes nesse bucket. Mas

para o caso da "features"quer-se que essa infraestrutura seja toda deletada depois de implementá-la em desenvolvimento. A solução encontrada foi adicionar dentro do workflow a troca desta configuração. Por consequência, antes de passar a template para implementação ele substitui a palavra "Retain"por "Delete", garantindo assim que vai ser deletado quando solicitado.

5.4 INTEGRAÇÃO E PERMISSÕES GITHUB- AWS

Um ponto importante são as permissões que foram criadas para a conta do GitHub que conseguem integrar com a conta da AWS. O método escolhido para fazer a conexão como comentado anteriormente foi utilizar "roles"e dentro da AWS considerar para esta role o GitHub como um provedor de identidade (idP), dando permissões e considerando o GitHub como um aplicativo seguro.

Para ser aceito ainda é preciso definir políticas de acesso, sendo uma delas mostrando de qual repositório e organização do GitHub está vindo o comando. Além disso consegue-se definir a branch, sendo que para a conta DEV, somente a branch "dev"e que comece com "feature"poderia fazer conexões. No caso da conta de produção somente a branch "main"pode fazer atualizações.

O AWS SAM e AWS cloudformation que utiliza esse role como permissão, precisa de autorizações especificas para criação e se necessário deletar os recursos criados. Assim é preciso verificar o que sistema vai criar e deletar e dar as permissões para cada recurso, evitando ao máximo dar permissões desnecessárias.

No caso da conta de produção, tem-se permissões de criar e deletar recursos como lambda ou Glue, mas os buckets são protegidos e não podem ser deletados via comando. Evita-se, assim, que em caso de algum ataque externo perca-se nossos dados.

Durante a utilização do sistema de idP, o GitHub fornece um thumbprint, um certificado usado para enviar chaves de acesso. Na criação da "role"o thumbprint é adicionado na conta da AWS. O problema é que o GiHub possui mais de um thumbprint, assim é importante verificar a documentação do GitHub e adicionar todos os thumbprints fornecidos.

6 RESULTADOS OBTIDOS

Para se analisar os resultados obtidos, precisamos voltar nos requisitos definidos, e verificar quais foram cumpridos, ou qual parte do requisito não foi cumprida e o motivo disso. Assim, neste capítulo vamos primeiro apresentar os requisitos funcionais, discutindo sobre como foi implementado e se o resultado foi o esperado.

Após realizada a análise dos requisitos funcionais, iremos verificar os requisitos não funcionais, discutindo o que foi implementado. Por ultimo iremos analisar os requisitos de negócios, podendo concluir se o projeto entregou o que foi pretendido

6.1 ANÁLISE REQUISITOS FUNCIONAIS

Os requisitos funcionais definidos foram cuidadosamente analisados em relação à implementação realizada. É importante ressaltar que esses requisitos foram estabelecidos como ponto de partida para a criação de um fluxo de CI/CD e a promoção de uma cultura de DevOps dentro da equipe. Ao revisar cada um dos requisitos, podemos constatar o seguinte:

Em relação à integração com um repositório do tipo Git (GitHub), o requisito foi totalmente atendido. O GitHub foi selecionado como o repositório principal, com políticas de uso e padronização bem definidas. Todos os códigos do projeto foram armazenados no repositório, permitindo o controle de versão eficiente.

No que diz respeito ao disparo automático de pipelines, é importante mencionar que, embora os workflows sejam acionados automaticamente quando ocorrem alterações no repositório, alguns processos ainda requerem ativação manual para evitar erros e permitir verificações de código. Essa abordagem foi adotada devido a restrições financeiras e de tempo, mas é altamente recomendável automatizar completamente o processo para garantir a agilidade e eficiência do pipeline. Assim, seria possível realizar todas as etapas automaticamente, desde a primeira modificação até a implantação em produção.

Quanto à implantação de serviços serverless, podemos afirmar com confiança que esse requisito foi plenamente cumprido. A migração da infraestrutura para código foi realizada com sucesso, utilizando o AWS CloudFormation e o AWS SAM. Essa abordagem permitiu definir e provisionar a infraestrutura de forma automatizada e consistente. A transformação da infraestrutura em código facilita a replicabilidade do sistema e simplifica a gerência de ambientes.

Por fim, o requisito de configuração de ambientes foi atendido por meio da implementação da infraestrutura como código. Com a divisão em dois ambientes distintos (desenvolvimento e produção) e a capacidade de realizar alterações independentes nos códigos e infraestrutura, foi possível cumprir esse requisito. A configuração de ambientes é essencial para garantir a estabilidade do sistema e facilitar o desenvolvimento

e teste de novas funcionalidades.

Concluindo, podemos afirmar que todos os requisitos funcionais foram cumpridos, com exceção da necessidade de automatizar completamente o processo de pipelines. Essa automação é fundamental para alcançar um fluxo de CI/CD mais eficiente e promover a cultura de DevOps de forma mais completa.

6.2 ANÁLISE REQUISITOS NÃO FUNCIONAIS

Os requisitos não funcionais estabelecidos foram avaliados cuidadosamente em relação à implementação realizada. Vamos examinar cada um deles de forma mais detalhada:

No que diz respeito ao desempenho, é importante ressaltar que não é possível realizar uma comparação direta com uma versão anterior. No entanto, alguns números indicam um bom desempenho do sistema atual. O tempo médio de implementação de uma nova infraestrutura é de aproximadamente 5 minutos, enquanto para alterações em infraestruturas existentes, o tempo médio é de cerca de 1 minuto. Esses tempos demonstram uma resposta rápida e eficiente do sistema durante o pipeline de CI/CD.

Para fornecer um contexto comparativo, vale mencionar que dentro da empresa temos outros pipelines que utilizam apenas a entrega contínua (CD) e empregam tecnologias como o Docker para a implementação de códigos com suas bibliotecas. Nos casos desses pipelines, os tempos para atualização variam entre 1 minuto e 3 minutos. Portanto, podemos confirmar que o tempo obtido no atual sistema está dentro do esperado e adequado em relação a outros processos semelhantes dentro da organização.

Em relação à segurança, podemos afirmar que o requisito foi plenamente atendido. Foram implementadas medidas para evitar a exclusão acidental dos buckets de produção, garantindo que apenas as permissões necessárias sejam concedidas. Comparado ao método anterior que utilizava permissões baseadas em usuários individuais, a abordagem atual utiliza permissões baseadas em funções (role-based IAM), o que torna o sistema mais seguro e restritivo.

Como discutido nos capítulos anteriores, nosso sistema foi configurado para permitir o acesso somente pela conta do GitHub e pelas branches designadas (main e dev). Essa abordagem reforça a segurança, pois limita o acesso somente a usuários autenticados e com as funções específicas atribuídas. Dessa forma, reduzimos significativamente as possibilidades de vulnerabilidades no sistema, garantindo que apenas usuários autorizados possam interagir com ele.

Quanto ao monitoramento, embora o GitHub Actions ofereça recursos de monitoramento e logs durante a execução dos workflows, não são enviadas notificações aos usuários finais para informá-los sobre o término dos pipelines ou a ocorrência de erros. Uma possível melhoria seria adicionar o envio de mensagens para um canal no

Slack, permitindo que os desenvolvedores recebam notificações sobre pull requests e falhas no pipeline. Essas notificações facilitariam o acompanhamento do progresso e a detecção rápida de problemas no processo de implantação.

Em relação à escalabilidade, o sistema demonstrou ser capaz de lidar com várias requisições simultâneas e de se expandir de forma eficiente. A infraestrutura como código permite a replicação e configuração fácil de ambientes adicionais, garantindo a capacidade do sistema de se adaptar às necessidades de crescimento e demanda.

Concluímos que a maioria dos requisitos não funcionais foi atendida de forma satisfatória. Destacamos o bom desempenho e a segurança do sistema, enquanto o monitoramento e a notificação de eventos importantes podem ser aprimorados para fornecer uma visão mais abrangente do status do pipeline.

6.3 ANÁLISE REQUISITOS DE NEGÓCIOS

O processo de CI/CD implementado busca garantir a conformidade com os padrões e políticas estabelecidos pela empresa. Diretrizes foram definidas no repositório do GitHub, como a utilização de branches específicas, nomenclatura de commits e organização de arquivos, a fim de promover a consistência e padronização do processo de desenvolvimento e implantação. Além disso, a infraestrutura como código permite a definição padronizada e reprodutível dos recursos, seguindo as melhores práticas estabelecidas. No entanto, é importante ressaltar que a conformidade completa depende da adesão e cumprimento dessas políticas por parte dos desenvolvedores e da equipe.

A automatização é um elemento fundamental no processo de CI/CD implementado. Os workflows do GitHub Actions são acionados automaticamente quando ocorrem alterações no repositório, permitindo a execução automatizada das etapas de compilação, teste e implantação. Isso promove agilidade e eficiência, possibilitando a implementação rápida de atualizações. No entanto, é importante destacar que ainda existem algumas etapas que requerem intervenção manual, como a aprovação de pull requests para envio de código às branches principais. Essa intervenção manual pode diminuir a velocidade do processo e não atingir o objetivo de automação completa. Sendo assim, é recomendado realizar revisões periódicas das etapas do processo de CI/CD, buscando identificar oportunidades de automação adicional e redução da intervenção manual.

No atual estágio do projeto, foram implementados testes principalmente relacionados à verificação de conformidade do código, como a aplicação de linters para garantir a consistência e qualidade do código fonte. No entanto, para um sistema de processamento de dados, é desejável ter testes adicionais que validem a qualidade e integridade dos dados processados. Esses testes podem incluir a verificação de formatos corretos, a detecção de valores inválidos ou ausentes, além da validação

de cálculos e transformações aplicadas aos dados. A implementação desses testes contribuirá para a confiabilidade do sistema, permitindo a detecção precoce de problemas ou anomalias nos dados, bem como a entrega de resultados mais consistentes e confiáveis para os usuários finais. Portanto, é recomendado considerar a inclusão de testes automatizados de qualidade de dados como parte do processo de CI/CD, a fim de garantir a qualidade e integridade dos dados processados pelo sistema.

7 CONCLUSÃO

A implementação da infraestrutura como código (IAC) foi a parte mais demorada, mas também a mais impactante. Sem ela, seria difícil construir um pipeline de CI/CD eficiente e replicar o ambiente de produção, incluindo a criação de ambientes de desenvolvimento. Agora, com a IAC, podemos replicar facilmente a infraestrutura para diferentes contas, permitindo a criação de novos ambientes, como ambientes de teste.

O primeiro objetivo desse projeto foi a organização da conta, e com a migração para a IAC, todos os recursos estão conectados às suas respectivas stacks, facilitando o monitoramento e obtenção de informações sobre eles por meio das "tags"atribuídas na AWS.

Quanto à implementação do CI/CD, conseguimos conectar o repositório e implementar automaticamente nas contas de desenvolvimento e produção. No entanto, existem etapas não automatizadas e poucos testes de software e qualidade de dados. A falta de implementação desses testes se deve, principalmente, a restrições de custo no momento. No entanto, o sistema está preparado para futuras implementações nesse sentido.

A conta de desenvolvimento foi fundamental para evitar o desenvolvimento direto na conta de produção e permitir testes em um ambiente isolado na AWS, minimizando o risco de falhas ou interrupções. Além disso, a conta de desenvolvimento possibilitou o monitoramento e análise separada dos custos de desenvolvimento e produção.

Além dos avanços tecnológicos e operacionais alcançados por meio deste projeto, é importante ressaltar os benefícios de qualidade de vida que ele traz para os desenvolvedores. A implementação da infraestrutura como código (IAC) proporcionou uma organização e padronização significativas, tornando muito mais fácil para a equipe entender como construir novos códigos e localizar os recursos relevantes. Com todas as stacks conectadas aos seus respectivos recursos, os desenvolvedores têm uma visão clara e estruturada do ambiente, reduzindo a complexidade e a incerteza ao desenvolver e fazer alterações.

Essa abordagem promove um ambiente de trabalho mais harmonioso e produtivo, permitindo que os desenvolvedores se concentrem em tarefas de maior valor, sem perder tempo navegando por uma infraestrutura caótica ou incerta. Como resultado, os desenvolvedores experimentam um aumento na eficiência e na satisfação profissional, contribuindo para uma cultura de colaboração e inovação contínua.

Em resumo, obtivemos resultados satisfatórios com este projeto, iniciando a cultura de DevOps para o processamento de dados no data lake. Identificamos práticas que se adequam e outras que necessitam de ajustes para o desenvolvimento com foco em processamento de dados. No entanto, também reconhecemos que ainda há muito a ser feito para alcançar um processo de CI/CD totalmente automatizado. Portanto,

este projeto é apenas o início de uma jornada contínua de aprimoramento.

REFERÊNCIAS

AGILE, State of. **The 14th Annual State of Agile Report**. [*S.l.*: *s.n.*]. https://info.digital.ai/rs/981-LQX-968/images/SOA14.pdf. [Acesso em: 26/06/2023].

ALOIA, João Victor Peria; RIBEIRO, José Eduardo. ANÁLISE COMPARATIVA ENTRE PROCESSOS DE ENTREGA DE SOFTWARE DO TRADICIONAL À ENTREGA CONTÍNUA. **Revista Científica Semana Acadêmica. Fortaleza**, v. 000153, 2018. https://semanaacademica.org.br/artigo/analise-comparativa-entre-processos-de-entrega-de-software-do-tradicional-entrega-continua.[Acesso em: 20/05/2023].

AMAZON WEB SERVICES, Inc. **Lambda programming model**. [*S.l.*: *s.n.*], 2023a. https://docs.aws.amazon.com/lambda/latest/dg/foundation-progmodel.html. [Acesso em: 25/06/2023].

AMAZON WEB SERVICES, Inc. **O que são microsserviços?** [*S.l.*: *s.n.*]. https://aws.amazon.com/pt/microservices/. [Acesso em: 12/05/2023].

AMAZON WEB SERVICES, Inc. **What is Amazon S3?** [*S.l.*: *s.n.*], 2023b. https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html. [Acesso em: 25/06/2023].

AMAZON WEB SERVICES, Inc. **What is AWS CloudFormation?** [*S.l.*: *s.n.*], 2023c. https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/Welcome.html. [Acesso em: 25/06/2023].

AMAZON WEB SERVICES, Inc. **What is AWS Glue?** [*S.l.*: *s.n.*], 2023d. https://docs.aws.amazon.com/glue/latest/dg/what-is-glue.html. [Acesso em: 25/06/2023].

AMAZON WEB SERVICES, Inc. **What is AWS Lambda?** [*S.l.*: *s.n.*], 2023e. https://docs.aws.amazon.com/lambda/latest/dg/welcome.html. [Acesso em: 25/06/2023].

AMAZON WEB SERVICES, Inc. **What is IAM??** [*S.l.*: *s.n.*], 2023f. https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html. [Acesso em: 25/06/2023].

AMAZON WEB SERVICES, Inc. **What is the AWS Serverless Application Model (AWS SAM)?** [*S.l.*: *s.n.*], 2023g. https://docs.aws.amazon.com/serverless-application-model/latest/developerguide/what-is-sam.html. [Acesso em: 25/06/2023].

AMAZON WEB SERVICES, Inc. **Working with nested stacks**. [*S.l.*: *s.n.*], 2023h. https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/using-cfn-nested-stacks.html. [Acesso em: 25/06/2023].

AMAZON WEB SERVICES, Inc. **Working with stacks**. [*S.l.*: *s.n.*], 2023i. https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/stacks.html. [Acesso em: 25/06/2023].

ATLASSIAN. **O que é Git**. [*S.l.*: *s.n.*]. https://www.atlassian.com/br/git/tutorials/what-is-git. [Acesso em: 25/05/2023].

AWS. **O que é DevOps?** [*S.l.*: *s.n.*]. https://aws.amazon.com/pt/devops/what-is-devops/. [Acesso em: 16/05/2023].

BUENO, Victor. **Diferenças entre Monólito e Microsserviços**. [*S.l.*: *s.n.*], 2022. https://dev.to/victorbueno/diferencas-entre-monolito-e-microsservicos-1n2g. [Acesso em: 28/05/2023].

CALDO, Carlos. **Data Quality: Como está a qualidade dos nossos dados?** [*S.l.*: *s.n.*], 2018. https://www.devmedia.com.br/data-quality-como-esta-a-qualidade-dos-nossos-dados/3021. [Acesso em: 25/06/2023].

CLOUD, Google. **O que é data lake?** [*S.l.*: *s.n.*]. https://cloud.google.com/learn/what-is-a-data-lake?hl=pt-br. [Acesso em: 12/05/2023].

DOROSZ, Lukasz. **CloudFormation i Stack Sets, łatwe zarządzanie na dużą skalę**. [*S.l.*: *s.n.*], 2019. https://lukado.eu/cloudformation-i-stack-sets-latwe-zarzadzanie-na-duza-skale/. [Acesso em: 30/05/2023].

FERNANDES, R. **Engenharia de Software**. [*S.l.*]: Curitiba: Fael., 2017.

GITHUB. **Entendendo o GitHub Actions**. [*S.l.*: *s.n.*]. `https://docs.github.com/pt/actions/learn-github-actions/understanding-github-actions`. [Acesso em: 25/05/2023].

GITHUB. **Sobre o Git**. [*S.l.*: *s.n.*]. `https://docs.github.com/pt/get-started/using-git/about-git`. [Acesso em: 25/05/2023].

HUMBLE, J.; FARLEY, D. **Entrega contínua: como entregar software de forma rápida e confiável**. [*S.l.*]: São Paulo: Bookman., 2013.

IMPACTA, Redação. **YAML: o que é e para que serve??** [*S.l.*: *s.n.*], 2021. `https://www.impacta.com.br/blog/yaml-o-que-e-e-para-que-serve/`. [Acesso em: 30/06/2023].

KANCZUK, Daniel. **O que são microsserviços e como funcionam?** [*S.l.*: *s.n.*], 2020. `https://blog.geekhunter.com.br/arquitetura-de-microsservicos-x-arquitetura-monolitica/`. [Acesso em: 28/05/2023].

LIMA, Jackson. **O que é infraestrutura como código?** [*S.l.*: *s.n.*], 2018. `https://blog.schoolofnet.com/o-que-e-infraestrutura-como-codigo/`. [Acesso em: 25/05/2023].

MARQUES, Rafael. **O que é Amazon AWS?** [*S.l.*: *s.n.*], 2023. `https://www.homehost.com.br/blog/tutoriais/o-que-e-amazon-aws/`. [Acesso em: 30/05/2023].

METZ, Cade. **Here's How Google Makes Sure It (Almost) Never Goes Down**. [*S.l.*: *s.n.*], 2016. `https://www.wired.com/2016/04/google-ensures-services-almost-never-go/`. [Acesso em: 18/05/2023].

MEZAK, Steve. **The Origins of DevOps: What's in a Name?** [*S.l.*: *s.n.*], 2018. `https://devops.com/the-origins-of-devops-whats-in-a-name/`. [Acesso em: 18/05/2023].

NAVIA, Diego. **Onboarding-DevOps**. [*S.l.*: *s.n.*], 2020. `https://github.com/Diegonavia/Onboarding-DevOps`. [Acesso em: 20/05/2023].

NOELLO, T. P. **Melhoria do processo de teste em uma empresa de desenvolvimento de software**. [*S.l.*: *s.n.*], 2016. `http://www.repositorio.jesuita.org.br/handle/UNISINOS/6126`. [Acesso em: 25/06/2023].

RESILLE, Willian. **Afinal, o que é DevOps?** [*S.l.*: *s.n.*], 2017. `https://agile.pub/assuntos-diversos/afinal-o-que-e-devops/`. [Acesso em: 19/05/2023].

ROBERTS, Mike. **Serverless Architectures**. [*S.l.*: *s.n.*], 2018. `https://martinfowler.com/articles/serverless.html`. [Acesso em: 28/05/2023].

SANOU, Bakary. **A Simplified Convention for Naming Branches and Commits in Git**. [*S.l.*: *s.n.*], 2022. `https://dev.to/varbsan/a-simplified-convention-for-naming-branches-and-commits-in-git-il4`. [Acesso em: 23/06/2023].

SANTOS, Wellington. **Porque usar Lint no seu projeto de testes**. [*S.l.*: *s.n.*], 2018. `https://imasters.com.br/back-end/porque-usar-lint-no-seu-projeto-de-testes`. [Acesso em: 25/06/2023].

SERGIO. **Scrum backlog: requisitos não funciona**. [*S.l.*: *s.n.*], 2014. `https://www.devmedia.com.br/scrum-backlog-requisitos-nao-funcionais/30203`. [Acesso em: 20/06/2023].

SHARMA, S. **DevOps for Dummies**. [*S.l.*]: IBM Limited Edition, John Wiley Sons, Inc., Hoboken., 2014.

Apêndices

APÊNDICE A – CÓDIGO DE IMPLEMENTAÇÃO DO CI/CD

Neste apêndice, são adicionados os dois códigos gerados para os workflows do GitHub Actions.

A.1 WORKFLOW DESENVOLVIDO PARA CONTA DE DESENVOLVIMENTO

```
name: Deploy AWS

on:
  push:
    branches:
      - 'dev'
      - 'feature**'

    paths:
      - '*/**'
      - 'MasterStack.yaml'
      - '!.github/workflows/**'
  delete:
    branches:
      - 'feature**'
env:
  AWS_REGION : ${{ secrets.REGION }}
  MASTER_STACK: MasterStack.yaml
  S3_BUCKET: stacks-file
  S3_BUCKET_FEATURE: stacks-feature
  AWS_ROLE: ${{ secrets.ROLE_DEV }}
permissions:
  id-token: write   # This is required for requesting the JWT
  contents: read    # This is required for actions/checkout

jobs:
  testes:
    if: github.event_name == 'push'
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - run: |
          # trigger the tests here
```

```
        echo 'Running unit tests or linter tests'
        pip3 install pytest
        echo 'Tests ok'
        ls

  delete-feature:
    if: startsWith(github.event.ref, 'feature') && github.
      event_name == 'delete'
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - uses: aws-actions/setup-sam@v2
        with:
          use-installer: true

      - name: configure aws credentials
        uses: aws-actions/configure-aws-credentials@v1.7.0
        with:
          role-to-assume: ${{ env.AWS_ROLE}} #change to reflect
              your IAM roles ARN
          role-session-name: GitHub_to_AWS
          aws-region: ${{ env.AWS_REGION }}
          role-duration-seconds: 3600
          role-skip-session-tagging: true

      - name: Delete feature branch stack
        env:
          FEATURE_BRANCH_NAME: ${{ github.event.ref }}
        run: |
          sam delete \
            --stack-name $(echo ${FEATURE_BRANCH_NAME##*/} | tr
                -cd '[a-zA-Z0-9-]') \
            --region ${AWS_REGION} \
            --no-prompts

  packaged-deploy-features:
    if: startsWith(github.ref, 'refs/heads/feature')
    needs: testes
    runs-on: ubuntu-latest
```

```
steps:

  - name: Checkout
    uses: actions/checkout@v3
    with:
      fetch-depth: 2
  - name: AWS
    uses: aws-actions/setup-sam@v2
    with:
      use-installer: true

      ref: ${{ github.event.pull_request.head.sha }}
  - name: PyPY
    uses: actions/setup-python@v4
    with:
      python-version: 'pypy3.9'

  - name: Set up AWS credentials
    env:
      AWS_ROLE: ${{ secrets.ROLE_STAGING }}

    uses: aws-actions/configure-aws-credentials@v1.7.0
    with:
      role-to-assume: ${{ env.AWS_ROLE}} #change to reflect
          your IAM roles ARN
      role-session-name: GitHub_to_AWS
      aws-region: ${{ env.AWS_REGION }}
      role-duration-seconds: 3600
      role-skip-session-tagging: true

  - name: Parse JSON
    run: |
      aws ssm get-parameter --region ${{ env.AWS_REGION }}
         --name /Roles/Feature/GitHub | jq -r '.Parameter.
         Value' > configuration.json
      echo "$(cat␣configuration.json)"
```

```
      - name: Chance Buckets Policies
        run: |
          sed -i 's/Retain/Delete/g' ${{ env.MASTER_STACK}}
      # Does deploy
      - name: Package and Deploy  Feature
        run: |

          sam package --template ${{ env.MASTER_STACK}}\
              --region us-west-2 \
              --s3-bucket ${{ env.S3_BUCKET_FEATURE}} \
              --output-template-file packaged-feature.yaml

          sam deploy --stack-name  $(echo ${GITHUB_REF##*/} |
             tr -cd '[a-zA-Z0-9-]') \
              --template packaged-feature.yaml\
              --s3-bucket ${{ env.S3_BUCKET_FEATURE}} \
              --capabilities CAPABILITY_AUTO_EXPAND \
              --region ${{ env.AWS_REGION }} \
              --no-fail-on-empty-changeset\
              --parameter-overrides $(cat configuration.json)


  packaged-deploy-DEV:
    if: github.ref == 'refs/heads/dev'
    needs: testes
    runs-on: ubuntu-latest


    steps:

      - name: Checkout
        uses: actions/checkout@v3
        with:
          fetch-depth: 2
      - name: AWS
        uses: aws-actions/setup-sam@v2
        with:
          use-installer: true
```

```
      ref: ${{ github.event.pull_request.head.sha }}
  - name: PyPY
    uses: actions/setup-python@v4
    with:
      python-version: 'pypy3.9'

  - name: Set up AWS credentials

    uses: aws-actions/configure-aws-credentials@v1.7.0
    with:
      role-to-assume: ${{ env.AWS_ROLE}} #change to reflect
          your IAM roles ARN
      role-session-name: GitHub_to_AWS
      aws-region: ${{ env.AWS_REGION }}
      role-duration-seconds: 3600
      role-skip-session-tagging: true

  - name: Parse  parameters JSON
    run: |
      aws ssm get-parameter --region ${{ env.AWS_REGION }}
         --name /Roles/Dev/GitHub | jq -r '.Parameter.Value
         ' > configuration.json
      echo "$(cat configuration.json)"



  # Does deploy Lambda
  - name: Package and Deploy DEV
    run: |

      sam package --template ${{ env.MASTER_STACK}}\
          --region us-west-2 \
          --s3-bucket ${{ env.S3_BUCKET}} \
          --output-template-file packaged-dev.yaml

      sam deploy --stack-name  MasterStack \
          --template packaged-dev.yaml\
          --s3-bucket ${{ env.S3_BUCKET}} \
```

```
            --capabilities CAPABILITY_AUTO_EXPAND \
            --region ${{ env.AWS_REGION }} \
            --no-fail-on-empty-changeset\
            --parameter-overrides $(cat configuration.json)
```

A.2 WORKFLOW DESENVOLVIDO PARA CONTA DE PRODUÇÃO

```
name: Deploy AWS prod

on:
  push:
    branches:
      - 'main'
    paths:
      - '*/**'
      - '!.github/workflows/**'
      - MasterStack.yaml

env:
  AWS_REGION : ${{ secrets.REGION }}
  MASTER_STACK: MasterStack.yaml
  S3_BUCKET: stacks-file
  S3_BUCKET_FEATURE: stacks-feature
  AWS_ROLE: ${{ secrets.ROLE_PRD }}
permissions:
  id-token: write   # This is required for requesting the JWT
  contents: read    # This is required for actions/checkout

jobs:
  testes:
    if: github.event_name == 'push'
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3
      - run: |
          # trigger the tests here
          echo 'Running unit tests or linter tests'
          pip3 install pytest
          echo 'Tests ok'
```

```
packaged-deploy-PRD:
  if: github.ref == 'refs/heads/main'
  needs: testes
  runs-on: ubuntu-latest


  steps:

    - name: Checkout
      uses: actions/checkout@v3
      with:
        fetch-depth: 2
    - name: AWS
      uses: aws-actions/setup-sam@v2
      with:
        use-installer: true

        ref: ${{ github.event.pull_request.head.sha }}
    - name: PyPY
      uses: actions/setup-python@v4
      with:
        python-version: 'pypy3.9'

    - name: Set up AWS credentials

      uses: aws-actions/configure-aws-credentials@v1.7.0
      with:
        role-to-assume: ${{ env.AWS_ROLE}}
        role-session-name: GitHub_to_AWS
        aws-region: ${{ env.AWS_REGION }}
        role-duration-seconds: 3600
        role-skip-session-tagging: true

    - name: Parse  parameters JSON
      run: |
        aws ssm get-parameter --region ${{ env.AWS_REGION }}
```

```
        --name /Roles/PRD/GitHub | jq -r '.Parameter.Value
        ' > configuration.json
      echo "$(cat configuration.json)"



- name: Package and Deploy PRD
  run: |

    sam package --template ${{ env.MASTER_STACK}}\
        --region ${{ env.AWS_REGION }} \
        --s3-bucket ${{ env.S3_BUCKET}} \
        --output-template-file packaged-PRD.yaml

    sam deploy --stack-name  MasterStack \
        --template packaged-PRD.yaml\
        --s3-bucket ${{ env.S3_BUCKET}} \
        --capabilities CAPABILITY_AUTO_EXPAND \
        --region ${{ env.AWS_REGION }} \
        --no-fail-on-empty-changeset\
        --parameter-overrides $(cat configuration.json)
```

APÊNDICE B – INFRAESTRUTURA COMO CÓDIGO DESENVOLVIDA

Neste apêndice, apresentamos uma coleção de códigos relacionados à Infraestrutura como Código (IAC).

B.1 MASTERSTACK

Apresentamos a MasterStack, a stack principal que constrói as demais.

```
AWSTemplateFormatVersion: "2010-09-09"
Transform: AWS::Serverless-2016-10-31

Parameters:
  RoleClean:
    Type: String
  RoleEnriched:
    Type: String
  RoleModels:
    Type: String
  RoleGlue:
    Type: String

Mappings:
  IsFeature:
    environment:
      test: "Delete"
      prod: "Retain"


Resources:

###################### STACKS ###################################
  CleanStack:
    Type: AWS::Serverless::Application
    Properties:
      Location: ./clean/CleanStack.yaml
      Parameters:
        BucketCleanName: !Ref BucketClean
        RoleClean: !Ref RoleClean
  EnrichedStack:
    Type: AWS::Serverless::Application
```

```
    Properties:
      Location: ./enriched/EnrichedStack.yaml
      Parameters:
        BucketRawName: !Ref BucketRaw
        BucketCleanName: !Ref  BucketClean
        BucketEnrichedName: !Ref BucketEnriched
        BucketGlueAssetsName: !Ref BucketGlueAssets
        RoleEnriched: !Ref RoleEnriched
        RoleGlue: !Ref RoleGlue

  ModelsStack:
    Type: AWS::Serverless::Application
    Properties:
      Location: ./models/ModelsStack.yaml
      Parameters:
        BucketCleanName: !Ref  BucketClean
        BucketEnrichedName: !Ref BucketEnriched
        BucketModelsName: !Ref BucketModels
        BucketGlueAssetsName: !Ref BucketGlueAssets
        RoleModels: !Ref RoleModels
        RoleGlue: !Ref RoleGlue




################## Bucket RAW #################
  BucketRaw:
    Type: AWS::S3::Bucket
    Properties:
      NotificationConfiguration:
        LambdaConfigurations:
          - Event: s3:ObjectCreated:Put
            Function: !GetAtt CleanStack.Outputs.CleanOLXArn
            Filter:
              S3Key:
                Rules:
                  - Name: prefix
                    Value: olx_listings/
                  - Name: suffix
```

```
            Value: .parquet
      - Event: s3:ObjectCreated:Put
        Function: !GetAtt CleanStack.Outputs.
          CleanVivarealArn
        Filter:
          S3Key:
            Rules:
              - Name: prefix
                Value: vivareal_listings/
              - Name: suffix
                Value: .parquet
      - Event: s3:ObjectCreated:Put
        Function: !GetAtt CleanStack.Outputs.
          CleanDetailsArn
        Filter:
          S3Key:
            Rules:
              - Name: prefix
                Value: listings_airbnbdetails/
              - Name: suffix
                Value: .parquet
      - Event: s3:ObjectCreated:Put
        Function: !GetAtt CleanStack.Outputs.
          CleanPriceAVArn
        Filter:
          S3Key:
            Rules:
              - Name: prefix
                Value: listings_airbnbpriceav/
              - Name: suffix
                Value: .parquet
      - Event: s3:ObjectCreated:Put
        Function: !GetAtt CleanStack.Outputs.
          CleanRankingLocationWeekendsArn
        Filter:
          S3Key:
            Rules:
              - Name: prefix
                Value: listings_ranking_location_weekends/
```

```
          - Name: suffix
            Value: .parquet
      - Event: s3:ObjectCreated:Put
        Function: !GetAtt CleanStack.Outputs.
          CleanRankingLocationArn
        Filter:
          S3Key:
            Rules:
              - Name: prefix
                Value: listings_ranking_location/
              - Name: suffix
                Value: .parquet
      - Event: s3:ObjectCreated:Put
        Function: !GetAtt CleanStack.Outputs.
          CleanRankingLocationAdultsArn
        Filter:
          S3Key:
            Rules:
              - Name: prefix
                Value: listings_ranking_location_adults/
              - Name: suffix
                Value: .parquet
      - Event: s3:ObjectCreated:Put
        Function: !GetAtt CleanStack.Outputs.
          CleanRankingLocationImportantDatesArn
        Filter:
          S3Key:
            Rules:
              - Name: prefix
                Value:
                  listings_ranking_location_important_dates
                  /
              - Name: suffix
                Value: .parquet
      - Event: s3:ObjectCreated:Put
        Function: !GetAtt CleanStack.Outputs.
          CleanLocationArn
        Filter:
          S3Key:
```

```
                Rules:
                  - Name: prefix
                    Value: listings_location/
                  - Name: suffix
                    Value: .parquet
    DeletionPolicy: Retain
    UpdateReplacePolicy: Retain



################## Bucket Clean #################
  BucketClean:
    Type: AWS::S3::Bucket
    DeletionPolicy: Retain
    UpdateReplacePolicy: Retain
################## Bucket enriched #################
  BucketEnriched:
    Type: AWS::S3::Bucket
    DeletionPolicy: Retain
    UpdateReplacePolicy: Retain

################## Bucket models #################
  BucketModels:
    Type: AWS::S3::Bucket
    DeletionPolicy: Retain
    UpdateReplacePolicy: Retain

################## Bucket Glue Assets #################
  BucketGlueAssets:
    Type: AWS::S3::Bucket
    DeletionPolicy: Retain
    UpdateReplacePolicy: Retain
```

B.2 CLEANSTACK

Apresentamos a CleanStack, a stack correspondente à camada "clean"(limpa) do Data Lake.

```
    AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31
```

```
Globals:
  Function:
    Handler: lambda_function.lambda_handler
    Runtime: python3.9
    MemorySize: 2048
    Timeout: 60
    Layers:
        - 'arn:aws:lambda:us-west-2:336392948345:layer:
          AWSDataWrangler-Python39:8'
    Tags:
        function: data_cleaning
        product: pipe

    Environment:
        Variables:
          BUCKET_OUTPUT: !Ref BucketCleanName

Parameters:
  BucketCleanName:
    Type: String
    Description: Bucket clean name
  RoleClean:
    Type: String
Resources:

################## LAMBDA FUNCTIONS ########################
  OLX:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: ./olx
      MemorySize: 10240
      Timeout: 600
      Role: !Ref RoleClean
      Events:
        SQSEvent:
          Type: SQS
          Properties:
```

```
            Queue: !GetAtt SQSQueueOLX.Arn
            BatchSize: 1
            Enabled: true

  Vivareal:
    Type: 'AWS::Serverless::Function'
    Properties:
      CodeUri: ./vivareal
      Timeout: 600
      Role: !Ref RoleClean
      Events:
          SQSEvent:
            Type: SQS
            Properties:
              Queue: !GetAtt SQSQueueVivareal.Arn
              BatchSize: 1
              Enabled: true

  Details:
    Type: 'AWS::Serverless::Function'
    Properties:
      CodeUri: ./details
      Timeout: 900
      Role: !Ref RoleClean
      Events:
          SQSEvent:
            Type: SQS
            Properties:
              Queue: !GetAtt SQSQueueDetails.Arn
              BatchSize: 1
              Enabled: true

  PriceAV:
    Type: 'AWS::Serverless::Function'
    Properties:
      CodeUri: ./price_av
      Timeout: 300
      Role: !Ref RoleClean
      Events:
```

```
        SQSEvent:
          Type: SQS
          Properties:
            Queue: !GetAtt SQSQueuePriceAV.Arn
            BatchSize: 1
            Enabled: true

  RankingLocationWeekends:
    Type: 'AWS::Serverless::Function'
    Properties:
      CodeUri: ./ranking_location_weekends
      Timeout: 120
      Role: !Ref RoleClean
      Events:
        SQSEvent:
          Type: SQS
          Properties:
            Queue: !GetAtt SQSQueueRankingLocationWeekends.
                Arn
            BatchSize: 1
            Enabled: true

  RankingLocation:
    Type: 'AWS::Serverless::Function'
    Properties:
      CodeUri: ./ranking_location
      MemorySize: 1024
      Timeout: 120
      Role: !Ref RoleClean
      Events:
        SQSEvent:
          Type: SQS
          Properties:
            Queue: !GetAtt SQSQueueRankingLocation.Arn
            BatchSize: 1
            Enabled: true

  RankingLocationAdults:
    Type: 'AWS::Serverless::Function'
```

```
    Properties:
      CodeUri: ./ranking_location_adults
      Timeout: 120
      Role: !Ref RoleClean
      Events:
          SQSEvent:
            Type: SQS
            Properties:
              Queue: !GetAtt SQSQueueRankingLocationAdults.Arn
              BatchSize: 1
              Enabled: true

  RankingLocationImportantDates:
    Type: 'AWS::Serverless::Function'
    Properties:
      CodeUri: ./ranking_location_important_dates
      MemorySize: 1024
      Timeout: 120
      Role: !Ref RoleClean
      Events:
          SQSEvent:
            Type: SQS
            Properties:
              Queue: !GetAtt
                  SQSQueueRankingLocationImportantDates.Arn
              BatchSize: 1
              Enabled: true

  Location:
    Type: 'AWS::Serverless::Function'
    Properties:
      Handler: lambda_function.lambda_handler
      Runtime: python3.8
      CodeUri: ./location
      MemorySize: 512
      Timeout: 900
      Role: !Ref RoleClean
      Events:
          SQSEvent:
```

```
            Type: SQS
            Properties:
              Queue: !GetAtt SQSQueueLocation.Arn
              BatchSize: 1
              Enabled: true

############################## S3 Invoke Permission
  ####################################
  LambdaInvokePermissionOLX:
    Type: 'AWS::Lambda::Permission'
    Properties:
      FunctionName: !GetAtt OLX.Arn
      Action: 'lambda:InvokeFunction'
      Principal: 's3.amazonaws.com'
      SourceAccount: !Sub ${AWS::AccountId}

  LambdaInvokePermissionVivareal:
    Type: 'AWS::Lambda::Permission'
    Properties:
      FunctionName: !GetAtt Vivareal.Arn
      Action: 'lambda:InvokeFunction'
      Principal: 's3.amazonaws.com'
      SourceAccount: !Sub ${AWS::AccountId}

  LambdaInvokePermissionDetails:
    Type: 'AWS::Lambda::Permission'
    Properties:
      FunctionName: !GetAtt Details.Arn
      Action: 'lambda:InvokeFunction'
      Principal: 's3.amazonaws.com'
      SourceAccount: !Sub ${AWS::AccountId}

  LambdaInvokePermissionPriceAV:
    Type: 'AWS::Lambda::Permission'
    Properties:
      FunctionName: !GetAtt PriceAV.Arn
      Action: 'lambda:InvokeFunction'
      Principal: 's3.amazonaws.com'
      SourceAccount: !Sub ${AWS::AccountId}
```

```
  LambdaInvokePermissionRankingLocationWeekends:
    Type: 'AWS::Lambda::Permission'
    Properties:
      FunctionName: !GetAtt RankingLocationWeekends.Arn
      Action: 'lambda:InvokeFunction'
      Principal: 's3.amazonaws.com'
      SourceAccount: !Sub ${AWS::AccountId}

  LambdaInvokePermissionRankingLocation:
    Type: 'AWS::Lambda::Permission'
    Properties:
      FunctionName: !GetAtt RankingLocation.Arn
      Action: 'lambda:InvokeFunction'
      Principal: 's3.amazonaws.com'
      SourceAccount: !Sub ${AWS::AccountId}

  LambdaInvokePermissionRankingLocationAdults:
    Type: 'AWS::Lambda::Permission'
    Properties:
      FunctionName: !GetAtt RankingLocationAdults.Arn
      Action: 'lambda:InvokeFunction'
      Principal: 's3.amazonaws.com'
      SourceAccount: !Sub ${AWS::AccountId}

  LambdaInvokePermissionRankingLocationImportantDates:
    Type: 'AWS::Lambda::Permission'
    Properties:
      FunctionName: !GetAtt RankingLocationImportantDates.Arn
      Action: 'lambda:InvokeFunction'
      Principal: 's3.amazonaws.com'
      SourceAccount: !Sub ${AWS::AccountId}

  LambdaInvokePermissionLocation:
    Type: 'AWS::Lambda::Permission'
    Properties:
      FunctionName: !GetAtt Location.Arn
      Action: 'lambda:InvokeFunction'
      Principal: 's3.amazonaws.com'
```

```
        SourceAccount: !Sub ${AWS::AccountId}

############################## SQS QUEUE
    ####################################
  SQSQueueOLX:
    Type: AWS::SQS::Queue
    Properties:
      FifoQueue: true
      VisibilityTimeout: 720

  SQSQueueVivareal:
    Type: AWS::SQS::Queue
    Properties:
      FifoQueue: true
      VisibilityTimeout: 720

  SQSQueueDetails:
    Type: AWS::SQS::Queue
    Properties:
      FifoQueue: true
      VisibilityTimeout: 960

  SQSQueuePriceAV:
    Type: AWS::SQS::Queue
    Properties:
      FifoQueue: true
      VisibilityTimeout: 420

  SQSQueueRankingLocationWeekends:
    Type: AWS::SQS::Queue
    Properties:
      FifoQueue: true
      VisibilityTimeout: 420

  SQSQueueRankingLocation:
    Type: AWS::SQS::Queue
    Properties:
      FifoQueue: true
      VisibilityTimeout: 240
```

```
  SQSQueueRankingLocationAdults:
    Type: AWS::SQS::Queue
    Properties:
      FifoQueue: true
      VisibilityTimeout: 240

  SQSQueueRankingLocationImportantDates:
    Type: AWS::SQS::Queue
    Properties:
      FifoQueue: true
      VisibilityTimeout: 240

  SQSQueueLocation:
    Type: AWS::SQS::Queue
    Properties:
      FifoQueue: true
      VisibilityTimeout: 960


############################## OUTPUTS
   ###################################
Outputs:
  CleanOLXArn:
    Value: !GetAtt OLX.Arn
  CleanVivarealArn:
    Value: !GetAtt Vivareal.Arn
  CleanDetailsArn:
    Value: !GetAtt Details.Arn
  CleanPriceAVArn:
    Value: !GetAtt PriceAV.Arn
  CleanRankingLocationWeekendsArn:
    Value: !GetAtt RankingLocationWeekends.Arn
  CleanRankingLocationArn:
    Value: !GetAtt RankingLocation.Arn
  CleanRankingLocationAdultsArn:
    Value: !GetAtt RankingLocationAdults.Arn
  CleanRankingLocationImportantDatesArn:
    Value: !GetAtt RankingLocationImportantDates.Arn
```

```
  CleanLocationArn:
    Value: !GetAtt Location.Arn
```

B.3 ENRICHEDSTACK

Apresentamos a EnrichedStack, a stack correspondente à camada "enriched"(enriquecido) do Data Lake.

```
    AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31

Parameters:
  BucketRawName:
    Type: String
  BucketCleanName:
    Type: String
    Description: Bucket clean name
  BucketEnrichedName:
    Type: String
    Description: Bucket enriched name
  BucketGlueAssetsName:
    Type: String
  RoleGlue:
    Type: String
  RoleEnriched:
    Type: String


Globals:
  Function:
    Handler: lambda_function.lambda_handler
    Runtime: python3.9
    MemorySize: 2048
    Timeout: 60
    Layers:
        - 'arn:aws:lambda:us-west-2:336392948345:layer:
          AWSDataWrangler-Python39:8'
    Tags:
        function: data_enrichment
        product: pipe
```

```
    Environment:
        Variables:
          BUCKET_CLEAN: !Ref BucketCleanName
          BUCKET_ENRICHED: !Ref BucketEnrichedName
Resources:
############################ LAMBDA FUNCTIONS
   ######################################

  DetailsLastAquisitionId:
    Type: 'AWS::Serverless::Function'
    Properties:
      CodeUri: ./details_last_aquisition_id
      Role: !Ref RoleEnriched
      MemorySize: 8192
      Timeout: 200
      Events:
        ScheduleDetailsLastAquisitionId:
          Type: Schedule
          Properties:
            Schedule: cron(30 18 ? * MON *)


  Terrenos:
    Type: 'AWS::Serverless::Function'
    Properties:
      CodeUri: ./terrenos
      Role: !Ref RoleEnriched
      Timeout: 330
      Events:
        ScheduleTerrenos:
          Type: Schedule
          Properties:
            Schedule: cron(0 18 ? * Tue *)

  ImoveisVenda:
    Type: 'AWS::Serverless::Function'
    Properties:
      CodeUri: ./imoveis_venda
      Role: !Ref RoleEnriched
```

```
      MemorySize: 5120
      Timeout: 885
      Events:
        ScheduleImoveisVenda:
          Type: Schedule
          Properties:
            Schedule: cron(0 18 ? * Tue *)

  ImoveisAluguel:
    Type: 'AWS::Serverless::Function'
    Properties:
      CodeUri: ./imoveis_aluguel
      Role: !Ref RoleEnriched
      MemorySize: 1024
      Timeout: 150
      Events:
        ScheduleImoveisAluguel:
          Type: Schedule
          Properties:
            Schedule: cron(0 18 ? * Tue *)

  ContatoCorretores:
    Type: 'AWS::Serverless::Function'
    Properties:
      CodeUri: ./contato_corretores
      Role: !Ref RoleEnriched
      MemorySize: 1024
      Timeout: 243
      Events:
        ScheduleContatoCorretores:
          Type: Schedule
          Properties:
            Schedule: cron(0 18 ? * Tue *)


############################ GLUE JOBS
   ######################################
  DeadAlive:
    Type: AWS::Glue::Job
```

```
    Properties:


      Role: !Ref RoleGlue
      Command:
        PythonVersion: 3
        ScriptLocation: ./dead_alive
        Name: glueetl
      WorkerType: G.1X
      GlueVersion: '3.0'
      NumberOfWorkers: 10
      MaxRetries: 0
      Timeout: 2880
      DefaultArguments:
        "--class": GlueApp
        "--CleanBucket": !Ref BucketCleanName
        "--EnrichedBucket": !Ref BucketEnrichedName
        "--enable-metrics": "true"
        "--enable-spark-ui": "true"
        "--spark-event-logs-path": !Sub "s3://${
           BucketGlueAssetsName}/sparkHistoryLogs/"
        "--enable-job-insights": "true"
        "--enable-glue-datacatalog": "true"
        "--enable-continuous-cloudwatch-log": "true"
        "--job-bookmark-option": "job-bookmark-enable"
        "--job-language": "python-3"
        "--TempDir" : !Sub "s3://${BucketGlueAssetsName}/
           temporary/"
      Tags:
          function: data_enrichment
          product: pipe

  HostStarRating:
      Type: AWS::Glue::Job
      Properties:
        Role: !Ref RoleGlue
        Command:
          PythonVersion: 3
          ScriptLocation: ./host_star_rating
```

```
      Name: glueetl
    WorkerType: G.1X
    NumberOfWorkers: 10
    GlueVersion: '3.0'
    MaxRetries: 0
    Timeout: 2880
    DefaultArguments:
      "--class": GlueApp
      "--CleanBucket": !Ref BucketCleanName
      "--EnrichedBucket": !Ref BucketEnrichedName
      "--enable-metrics": "true"
      "--enable-spark-ui": "true"
      "--spark-event-logs-path": !Sub "s3://${
        BucketGlueAssetsName}/sparkHistoryLogs/"
      "--enable-job-insights": "true"
      "--enable-glue-datacatalog": "true"
      "--enable-continuous-cloudwatch-log": "true"
      "--job-bookmark-option": "job-bookmark-enable"
      "--job-language": "python-3"
      "--TempDir" : !Sub "s3://${BucketGlueAssetsName}/
        temporary/"
    Tags:
        function: data_enrichment
        product: pipe

BookedOn:
  Type: AWS::Glue::Job
  Properties:
    Role: !Ref RoleGlue
    Command:
      PythonVersion: 3
      ScriptLocation: ./booked_on
      Name: glueetl
    WorkerType: G.2X
    NumberOfWorkers: 15
    GlueVersion: '4.0'
    MaxRetries: 0
    Timeout: 300
    DefaultArguments:
```

```
        "--class": GlueApp
        "--CleanBucket": !Ref BucketCleanName
        "--EnrichedBucket": !Ref BucketEnrichedName
        "--enable-metrics": "true"
        "--enable-spark-ui": "false"
        "--write-shuffle-files-to-s3": "true"
        "--spark-event-logs-path": !Sub "s3://${
           BucketGlueAssetsName}/sparkHistoryLogs/"
        "--enable-job-insights": "false"
        "--enable-glue-datacatalog": "true"
        "--enable-continuous-cloudwatch-log": "true"
        "--job-bookmark-option": "job-bookmark-disable"
        "--job-language": "python-3"
        "--TempDir" : !Sub "s3://${BucketGlueAssetsName}/
           temporary/booked_on/"
      Tags:
          function: data_enrichment
          product: pipe


BlockAndOccupancy:
    Type: AWS::Glue::Job
    Properties:
      Role: !Ref RoleGlue
      Command:
        PythonVersion: 3
        ScriptLocation: ./block_and_occupancy
        Name: glueetl
      WorkerType: G.2X
      NumberOfWorkers: 30
      GlueVersion: '4.0'
      MaxRetries: 0
      Timeout: 300
      DefaultArguments:
        "--class": GlueApp
        "--CleanBucket": !Ref BucketCleanName
        "--EnrichedBucket": !Ref BucketEnrichedName
        "--enable-metrics": "true"
        "--enable-spark-ui": "true"
```

```
        "--write-shuffle-files-to-s3": "true"
        "--spark-event-logs-path": !Sub "s3://${
            BucketGlueAssetsName}/sparkHistoryLogs/BaO/"
        "--enable-job-insights": "true"
        "--enable-glue-datacatalog": "true"
        "--enable-continuous-cloudwatch-log": "true"
        "--job-bookmark-option": "job-bookmark-disable"
        "--job-language": "python-3"
        "--TempDir" : !Sub "s3://${BucketGlueAssetsName}/
            temporary/BaO/"
      Tags:
          function: data_enrichment
          product: pipe

Reservetions:
    Type: AWS::Glue::Job
    Properties:
      Role: !Ref RoleGlue
      Command:
        PythonVersion: 3
        ScriptLocation: ./reservations
        Name: glueetl
      WorkerType: G.1X
      NumberOfWorkers: 10
      GlueVersion: '3.0'
      MaxRetries: 0
      Timeout: 300
      DefaultArguments:
        "--class": GlueApp
        "--CleanBucket": !Ref BucketCleanName
        "--EnrichedBucket": !Ref BucketEnrichedName
        "--enable-metrics": "true"
        "--enable-spark-ui": "false"
        "--spark-event-logs-path": !Sub "s3://${
            BucketGlueAssetsName}/sparkHistoryLogs/
            reservations"
        "--enable-job-insights": "true"
        "--enable-glue-datacatalog": "true"
        "--enable-continuous-cloudwatch-log": "true"
```

```
        "--job-bookmark-option": "job-bookmark-enable"
        "--job-language": "python-3"
        "--TempDir" : !Sub "s3://${BucketGlueAssetsName}/
            temporary/reservations"
      Tags:
          function: data_enrichment
          product: pipe


  DailyFat:
      Type: AWS::Glue::Job
      Properties:
        Role: !Ref RoleGlue
        Command:
          PythonVersion: 3
          ScriptLocation: ./daily_fat
          Name: glueetl
        WorkerType: G.1X
        NumberOfWorkers: 10
        GlueVersion: '3.0'
        MaxRetries: 0
        Timeout: 300
        DefaultArguments:
          "--class": GlueApp
          "--CleanBucket": !Ref BucketCleanName
          "--EnrichedBucket": !Ref BucketEnrichedName
          "--enable-metrics": "true"
          "--enable-spark-ui": "true"
          "--spark-event-logs-path": !Sub "s3://${
              BucketGlueAssetsName}/sparkHistoryLogs/"
          "--enable-job-insights": "true"
          "--enable-glue-datacatalog": "true"
          "--enable-continuous-cloudwatch-log": "true"
          "--job-bookmark-option": "job-bookmark-enable"
          "--job-language": "python-3"
          "--TempDir" : !Sub "s3://${BucketGlueAssetsName}/
              temporary/"
        Tags:
            function: data_enrichment
```

```
            product: pipe

  CleaningBlockTaggedReservations:
      Type: AWS::Glue::Job
      Properties:
        Role: !Ref RoleGlue
        Command:
          PythonVersion: 3
          ScriptLocation: ./cleaning_block_tagged_reservations
          Name: glueetl
        WorkerType: G.1X
        NumberOfWorkers: 10
        GlueVersion: '3.0'
        MaxRetries: 0
        Timeout: 2880
        DefaultArguments:
          "--class": GlueApp
          "--CleanBucket": !Ref BucketCleanName
          "--EnrichedBucket": !Ref BucketEnrichedName
          "--enable-metrics": "true"
          "--enable-spark-ui": "true"
          "--spark-event-logs-path": !Sub "s3://${
              BucketGlueAssetsName}/sparkHistoryLogs/"
          "--enable-job-insights": "true"
          "--enable-glue-datacatalog": "true"
          "--enable-continuous-cloudwatch-log": "true"
          "--job-bookmark-option": "job-bookmark-disable"
          "--job-language": "python-3"
          "--TempDir" : !Sub "s3://${BucketGlueAssetsName}/
              temporary"
        Tags:
            function: data_enrichment
            product: pipe

  DiscountReservations:
      Type: AWS::Glue::Job
      Properties:
        Role: !Ref RoleGlue
        Command:
```

```
        PythonVersion: 3
        ScriptLocation: ./discounts_reservations
      Name: glueetl
    WorkerType: G.1X
    NumberOfWorkers: 5
    GlueVersion: '3.0'
    MaxRetries: 0
    Timeout: 300
    DefaultArguments:
      "--class": GlueApp
      "--RawBucket": !Ref BucketRawName
      "--EnrichedBucket": !Ref BucketEnrichedName
      "--enable-metrics": "true"
      "--enable-spark-ui": "true"
      "--spark-event-logs-path": !Sub "s3://${
        BucketGlueAssetsName}/sparkHistoryLogs/"
      "--enable-job-insights": "true"
      "--enable-glue-datacatalog": "true"
      "--enable-continuous-cloudwatch-log": "true"
      "--job-bookmark-option": "job-bookmark-disable"
      "--job-language": "python-3"
      "--TempDir" : !Sub "s3://${BucketGlueAssetsName}/
        temporary"
    Tags:
        function: data_enrichment
        product: pipe

CleaningBlockReservationModifer:
  Type: AWS::Glue::Job
  Properties:
    Role: !Ref RoleGlue
    Command:
      PythonVersion: 3
      ScriptLocation: ./cleaning_block_reservation_modifier
      Name: glueetl
    WorkerType: G.1X
    NumberOfWorkers: 10
    GlueVersion: '3.0'
    MaxRetries: 0
```

```
      Timeout: 2880
      DefaultArguments:
        "--class": GlueApp
        "--RawBucket": !Ref BucketRawName
        "--CleanBucket": !Ref BucketCleanName
        "--EnrichedBucket": !Ref BucketEnrichedName
        "--enable-metrics": "true"
        "--enable-spark-ui": "true"
        "--spark-event-logs-path": !Sub "s3://${
           BucketGlueAssetsName}/sparkHistoryLogs/"
        "--enable-job-insights": "true"
        "--enable-glue-datacatalog": "true"
        "--enable-continuous-cloudwatch-log": "true"
        "--job-bookmark-option": "job-bookmark-disable"
        "--job-language": "python-3"
        "--TempDir" : !Sub "s3://${BucketGlueAssetsName}/
           temporary"
      Tags:
          function: data_enrichment
          product: pipe

  MonthlyFat:
    Type: AWS::Glue::Job
    Properties:
      Role: !Ref RoleGlue
      Command:
        PythonVersion: 3
        ScriptLocation: ./monthly_fat
        Name: glueetl
      WorkerType: G.1X
      NumberOfWorkers: 10
      GlueVersion: '3.0'
      MaxRetries: 0
      Timeout: 300
      DefaultArguments:
        "--class": GlueApp
        "--EnrichedBucket": !Ref BucketEnrichedName
        "--enable-metrics": "true"
        "--enable-spark-ui": "true"
```

```
        "--spark-event-logs-path": !Sub "s3://${
            BucketGlueAssetsName}/sparkHistoryLogs/"
        "--enable-job-insights": "true"
        "--enable-glue-datacatalog": "true"
        "--enable-continuous-cloudwatch-log": "true"
        "--job-bookmark-option": "job-bookmark-enable"
        "--job-language": "python-3"
        "--TempDir" : !Sub "s3://${BucketGlueAssetsName}/
            temporary"
      Tags:
          function: data_enrichment
          product: pipe
```

B.4 MODELSSTACK

Apresentamos a ModelsStack, a stack correspondente à camada "models"(modelos) do Data Lake.

```
    AWSTemplateFormatVersion: '2010-09-09'
Transform: AWS::Serverless-2016-10-31

Parameters:
  BucketCleanName:
    Type: String
    Description: Bucket clean name
  BucketEnrichedName:
    Type: String
    Description: Bucket enriched name
  BucketModelsName:
    Type: String
  BucketGlueAssetsName:
    Type: String
  RoleGlue:
    Type: String
  RoleModels:
    Type: String



Globals:
```

```
  Function:
    Handler: lambda_function.lambda_handler
    Runtime: python3.9
    MemorySize: 2048
    Timeout: 60
    Layers:
        - 'arn:aws:lambda:us-west-2:336392948345:layer:
          AWSDataWrangler-Python39:8'
    Tags:
        function: data_enrichment
        product: pipe
    Environment:
        Variables:
          BUCKET_CLEAN: !Ref BucketCleanName
          BUCKET_ENRICHED: !Ref BucketEnrichedName
          BUCKET_MODELS: !Ref BucketModelsName

Resources:

############################ LAMBDA FUNCTIONS
   ######################################

  SemaforoMonthlyRevenue:
    Type: 'AWS::Serverless::Function'
    Properties:
      CodeUri: ./semaforo_monthly_revenue
      Role: !Ref RoleModels
      MemorySize: 1000
      Timeout: 300
      Events:
        ScheduleSemaforoMonthlyRevenue:
          Type: Schedule
          Properties:
            Schedule: cron(30 21 ? * MON *)
############################ GLUE JOBS
   ######################################
  RevenueModels:
    Type: AWS::Glue::Job
    Properties:
```

```
      Role: !Ref RoleGlue
      Command:
        PythonVersion: 3
        ScriptLocation: ./revenue_models
        Name: glueetl
      WorkerType: G.2X
      GlueVersion: '3.0'
      NumberOfWorkers: 15
      MaxRetries: 0
      Timeout: 2880
      DefaultArguments:
        "--class": GlueApp
        "--CleanBucket": !Ref BucketCleanName
        "--EnrichedBucket": !Ref BucketEnrichedName
        "--enable-metrics": "true"
        "--enable-spark-ui": "true"
        "--additional-python-modules": "statsmodels==0.13.2"
        "--conf": "spark.executor.memory=14g"
        "--write-shuffle-files-to-s3": "true"
        "--write-shuffle-spills-to-s3": "true"
        "--spark-event-logs-path": !Sub "s3://${
          BucketGlueAssetsName}/sparkHistoryLogs/"
        "--enable-job-insights": "true"
        "--enable-glue-datacatalog": "true"
        "--enable-continuous-cloudwatch-log": "true"
        "--job-bookmark-option": "job-bookmark-enable"
        "--job-language": "python-3"
        "--TempDir" : !Sub "s3://${BucketGlueAssetsName}/
          temporary/"
      Tags:
          function: data_modeling
          product: pipe

  MonthlyRevenueModels:
    Type: AWS::Glue::Job
    Properties:
      Role: !Ref RoleGlue
      Command:
        PythonVersion: 3
```

```
      ScriptLocation: ./monthly_revenue_models
      Name: glueetl
    WorkerType: G.2X
    GlueVersion: '3.0'
    NumberOfWorkers: 6
    MaxRetries: 0
    Timeout: 2880
    DefaultArguments:
      "--class": GlueApp
      "--CleanBucket": !Ref BucketCleanName
      "--EnrichedBucket": !Ref BucketEnrichedName
      "--enable-metrics": "true"
      "--enable-spark-ui": "true"
      "--conf": "spark.executor.memory=24g"
      "--write-shuffle-files-to-s3": "true"
      "--write-shuffle-spills-to-s3": "true"
      "--spark-event-logs-path": !Sub "s3://${
        BucketGlueAssetsName}/sparkHistoryLogs/"
      "--enable-job-insights": "true"
      "--enable-glue-datacatalog": "true"
      "--enable-continuous-cloudwatch-log": "true"
      "--job-bookmark-option": "job-bookmark-enable"
      "--job-language": "python-3"
      "--TempDir" : !Sub "s3://${BucketGlueAssetsName}/
        temporary/"
    Tags:
        function: data_modeling
        product: pipe


MonthlyRevenueModelsByYear:
  Type: AWS::Glue::Job
  Properties:
    Role: !Ref RoleGlue
    Command:
      PythonVersion: 3
      ScriptLocation: ./monthly_revenue_models_by_year
      Name: glueetl
    WorkerType: G.2X
```

```
      GlueVersion: '3.0'
      NumberOfWorkers: 6
      MaxRetries: 0
      Timeout: 2880
      DefaultArguments:
        "--class": GlueApp
        "--CleanBucket": !Ref BucketCleanName
        "--EnrichedBucket": !Ref BucketEnrichedName
        "--enable-metrics": "true"
        "--enable-spark-ui": "true"
        "--conf": "spark.executor.memory=24g"
        "--write-shuffle-files-to-s3": "true"
        "--write-shuffle-spills-to-s3": "true"
        "--spark-event-logs-path": !Sub "s3://${
           BucketGlueAssetsName}/sparkHistoryLogs/monthly-
           revenue-models/"
        "--enable-job-insights": "true"
        "--enable-glue-datacatalog": "true"
        "--enable-continuous-cloudwatch-log": "true"
        "--job-bookmark-option": "job-bookmark-enable"
        "--job-language": "python-3"
        "--TempDir" : !Sub "s3://${BucketGlueAssetsName}/
           temporary/"
      Tags:
          function: data_modeling
          product: pipe

  MonthlyOccupancyModels:
    Type: AWS::Glue::Job
    Properties:
      Role: !Ref RoleGlue
      Command:
        PythonVersion: 3
        ScriptLocation: ./monthly_occupancy_models
        Name: glueetl
      WorkerType: G.2X
      GlueVersion: '3.0'
      NumberOfWorkers: 6
      MaxRetries: 0
```

```
Timeout: 2880
DefaultArguments:
  "--class": GlueApp
  "--CleanBucket": !Ref BucketCleanName
  "--EnrichedBucket": !Ref BucketEnrichedName
  "--enable-metrics": "true"
  "--enable-spark-ui": "true"
  "--conf": "spark.executor.memory=24g"
  "--write-shuffle-files-to-s3": "true"
  "--write-shuffle-spills-to-s3": "true"
  "--spark-event-logs-path": !Sub "s3://${
     BucketGlueAssetsName}/sparkHistoryLogs/monthly-
     revenue-models/"
  "--enable-job-insights": "true"
  "--enable-glue-datacatalog": "true"
  "--enable-continuous-cloudwatch-log": "true"
  "--job-bookmark-option": "job-bookmark-enable"
  "--job-language": "python-3"
  "--TempDir" : !Sub "s3://${BucketGlueAssetsName}/
     temporary/"
Tags:
    function: data_modeling
    product: pipe
```