

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Matheus Da Silva Vieira

**Desenvolvimento de Aplicações Utilizando
DDD e Cloud Computing**

Uberlândia, Brasil

2023

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

Matheus Da Silva Vieira

Desenvolvimento de Aplicações Utilizando DDD e Cloud Computing

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Orientador: Ronaldo Castro de Oliveira

Universidade Federal de Uberlândia – UFU

Faculdade de Computação

Bacharelado em Sistemas de Informação

Uberlândia, Brasil

2023

Matheus Da Silva Vieira

Desenvolvimento de Aplicações Utilizando DDD e Cloud Computing

Trabalho de conclusão de curso apresentado à Faculdade de Computação da Universidade Federal de Uberlândia, como parte dos requisitos exigidos para a obtenção título de Bacharel em Sistemas de Informação.

Trabalho aprovado. Uberlândia, Brasil, 08 de Fevereiro de 2023:

Ronaldo Castro de Oliveira
Orientador

Mauricio Cunha Escarpinati

Daniel Duarte Abdala

Uberlândia, Brasil
2023

Dedico ao professor Ronaldo Castro de Oliveira que esteve presente ao meu lado me apoiando e incentivando no desenvolvimento deste trabalho.

Agradecimentos

Primeiramente à Deus, quem me concedeu folego de vida, me deu o privilégio de construir esse trabalho e se fez presente na minha caminhada.

Minha mãe, Maria Glória, meu irmão, Luiz Gustavo, que foram primordiais em toda a minha caminhada desde o ingresso até a conclusão.

Meu primo, Thiago Sol que me acompanhou durante a graduação com grande apoio .

Minha namorada, Juliana Eugênia, por todo carinho, apoio e incentivo.

Todos que influenciaram diretamente ou indiretamente na minha formação.

Ao corpo docente da Universidade Federal de Uberlândia, coordenação e colegiado que foram essenciais no processo da graduação.

Resumo

Para aplicar os conceitos do *Domain-Driven Design* no desenvolvimento de software e criar toda a infraestrutura necessária para disponibilizar esta solução na nuvem. Foram propostos requisitos no formato de casos de uso de um sistema, demonstrando que, através dos pilares do *DDD* é possível separar domínios, utilizar boas práticas de desenvolvimento, definir uma linguagem comum à todos que fazem parte do sistema e construir um design orientado por modelos. Com a arquitetura proposta, foram criados recursos necessários de infraestrutura, através de código, para ser aplicados na nuvem, possibilitando realizar a entrega das aplicações, utilizando imagens *Dockers*. E então, toda configuração de integração, permissão e rede de acesso, foram provisionadas através de códigos utilizando a linguagem *Terraform*.

Palavras-chave: Domain-Driven Design, Cloud Computing, Desenvolvimento, Provedores de Nuvem.

Lista de ilustrações

Figura 1 – Visão Geral da Linguagem de Padrões (ERIC EVANS, 2016)	14
Figura 2 – Fluxo de trabalho do Elastic Beanstalk. (AWS-BENSTALK, 2022) . .	20
Figura 3 – Características do DynamoDB. (AWS-DYNAMODB, 2022)	21
Figura 4 – Elastic Load Balancing.	22
Figura 5 – Amazon Simple Storage Service (Amazon S3). (AWS-S3, 2022)	23
Figura 6 – Amazon Simple Notification Service. (AWS-SNS, 2022)	24
Figura 7 – Amazon Simple Queue Service. (AWS-SQS, 2022)	25
Figura 8 – Amazon Relational Database Service (AWS-RDS, 2022)	25
Figura 9 – Modelagem Estratégica	28
Figura 10 – Mapa de Contexto	29
Figura 11 – Arquitetura Contextual	30
Figura 12 – Entidade Cliente	31
Figura 13 – Documento Endereço	31
Figura 14 – API de Cliente	32
Figura 15 – Pacote Infraestrutura cliente-api.	33
Figura 16 – Pacote Casos de Uso cliente-api.	34
Figura 17 – Serviço de Operações	35
Figura 18 – Interface da Aplicação.	36
Figura 19 – Infraestrutura como Código	37
Figura 20 – Aplicações no Elastic Beanstalk	38
Figura 21 – Tabela do DynamoDB	38
Figura 22 – Documentos do Cliente	39
Figura 23 – Banco de Dados Operação.	39
Figura 24 – Instâncias em execução.	40
Figura 25 – Página de Cadastro.	41
Figura 26 – Página Inicial da Interface.	42
Figura 27 – Página de Busca.	43
Figura 28 – Página de Atualizar Cliente.	44
Figura 29 – Página de Situação do Cliente.	45
Figura 30 – Página de Busca de Operações do Cliente.	46
Figura 31 – Arquitetura Projeto de Infraestrutura.	52
Figura 32 – Arquitetura Da Aplicações.	53
Figura 33 – Arquitetura Pacote Infra.	54

Sumário

1	INTRODUÇÃO	9
1.1	Objetivos Gerais e Específicos	10
1.2	Trabalhos Correlatos	10
1.3	Organização do Trabalho	12
2	DOMAIN-DRIVEN DESIGN	13
2.1	Definição	13
2.2	Pilares	13
2.2.1	Linguagem Ubíqua	13
2.2.2	Contexto Delimitado	15
2.2.3	Mapa de Contexto	15
2.2.4	Design Orientado por Modelos	15
2.2.5	Arquitetura em Camadas	16
2.2.6	Entidades e Agregadores	16
2.2.7	Objetos de Valor	16
2.2.8	Repositórios	17
2.2.9	Serviços	17
3	CLOUD COMPUTING	18
3.1	Definição	18
3.2	Computação em Nuvem Como Serviço	18
3.3	Software como um serviço	18
3.4	Plataforma como um serviço	19
3.5	Infraestrutura como um serviço	19
3.6	Provedor de nuvem Amazon Web Services (AWS)	19
3.6.1	AWS Elastic Beanstalk	19
3.6.2	Amazon Elastic Compute Cloud	20
3.6.3	Amazon DynamoDB	21
3.6.4	Elastic Load Balancing	22
3.6.5	Amazon S3	22
3.6.6	Amazon Simple Notification Service	23
3.6.7	Amazon Simple Queue Service	24
3.6.8	Amazon RDS	25
4	DESENVOLVIMENTO	27
4.1	Aplicação da modelagem utilizando conceito do DDD	27

4.1.1	Glossário	27
4.1.2	Contexto Delimitado em Caso de Uso	27
4.1.3	Mapa de Contextos	28
4.1.4	Arquitetura Contextual	29
4.2	Implementando a aplicação de Cliente	30
4.2.1	Arquitetura do Projeto	32
4.3	Construção do componente de Operações	35
4.4	Interface do Sistema	36
4.5	Disponibilizando os Recursos na Nuvem	37
4.6	Executando aplicação com a Interface	40
5	CONCLUSÃO	47
	REFERÊNCIAS	49
	ANEXO A – PROJETO DE INFRAESTRUTURA COMO CÓDIGO	52
	ANEXO B – REPOSITÓRIOS PARA COLABORAÇÃO	55

1 Introdução

Com a evolução diária da tecnologia e os sistemas que a acompanham, está cada vez mais fácil e rápido o desenvolvimento e publicação de softwares em produção, porém podem ocorrer dificuldades na manutenção e escalabilidade desses. A facilidade em desenvolvê-los também é consequência de ferramentas que auxiliam na criação do projeto e desenvolvimento dos mesmos, ou seja, a configuração inicial já é gerada através de plataformas que disponibilizam uma *CLI* (Command-line Interface) (BRIAN W. KERNIGHAN; ROB PIKE, 1984) que constrói o projeto com base nas informações dadas. Algumas ferramentas dão a opção de utilizar uma *API* (Application Programming Interface) (COTTON IW; GREATOREX FS; JR, 1968) gráfica para fornecer de maneira mais intuitiva essa criação e *frameworks* (STÉPHANE NICOLL; DAVE SYER; MADHURA BHAVE, 2022) que contêm abstrações de códigos já funcionais e testados, que ajudam bastante na codificação do software. A rapidez em publicar e disponibilizar sistemas em produção também é consequência de automações feitas na infraestrutura, onde com poucos cliques é possível realizar a entrega do código e disponibilizar no ambiente.

Grande parte das dificuldades que são encontradas na construção de um software podem estar relacionadas à modelagem, definição de domínio, separação de contexto e linguagem ubíqua. Esses termos são relatados por ERIC EVANS (2016) na escrita do *DDD* (Domain-Driven Design) que especifica uma maneira de escrever software orientado a domínio.

Os modelos tradicionais, como por exemplo o *Model-view-controller*, que foi amplamente adotado pela indústria de desenvolvimento web. Este modelo, direciona o desenvolvimento para implementação e não para o domínio do software. Com isso, o software fica vulnerável a *code smells*, ou seja, más práticas de desenvolvimento. De acordo com o artigo do (ANICHE et al., 2018), a arquitetura *MVC*, pode gerar muito controle de fluxo nas classes controladoras, métodos com várias ações e responsabilidades, repositório gerenciando muitas entidades, com lógicas complexas e classes de serviços consultando diretamente o banco de dados.

Ao contrário dos modelos tradicionais, o *DDD* define uma linguagem ubíqua a ser praticada com base na separação de contexto com alguns princípios com foco no domínio, explorando novos modelos de forma criativa. Por meio dos pilares do *DDD* consegue-se escrever e modelar sistemas de modo que, seja possível expandir o negócio respeitando as separações de domínio entre os sistemas (EVANS, 2015) (RADEMACHER; SORGALLA; SACHWEH, 2018a).

Para possibilitar a expansão dos sistemas distribuídos de maneira mais simples e

prática, a fim de colocar mais softwares em produção, a computação em nuvem ajuda a entregar e manter recursos de tecnologia como: banco de dados, aplicações, integrações, configurações, escalabilidade e segurança. Ofertando os recursos sob demanda da internet com preços tabelados de acordo com o uso. Ao invés de adquirir servidores de grande porte, redes e fazer a manutenção de *Data Centers*, os provedores de nuvem oferecem esses serviços sob demanda. Então, ao contratar um provedor desse, é possível economizar custos, pois não é preciso ter servidores físicos conectados o tempo todo para garantir a disponibilidade, ter elasticidade para tratar picos de requisições, podendo aumentar e diminuir recursos instantaneamente e ser ágil ao criar e destruir componentes da infraestrutura.(AWS, 2022a)

Portanto o *Domain-Driven Design* é uma abordagem para o desenvolvimento de softwares complexos onde o objetivo é concentrar no domínio principal, explorar modelos em uma colaboração criativa entre os profissionais de domínio e falar a linguagem ubíqua dentro de um contexto limitado explicitamente. E a computação em nuvem possibilita a utilização de recursos computacionais sob demanda e modelos de serviços que possam nos atender. Portanto, este trabalho irá abordar conceitos e caso de uso aplicando *DDD* em *Cloud Computing*.

1.1 Objetivos Gerais e Específicos

O objetivo geral desse trabalho de conclusão de curso é aplicar conceitos do *Domain-Driven Design* no desenvolvimento de software e criar toda a infraestrutura necessária para disponibilizar esta solução na nuvem.

Para isto, foram abordados objetivos específicos como os conceitos e boas práticas do *DDD*, com o objetivo de serem aplicados nos requisitos propostos. Resultando em modelagem e arquitetura contextual da solução. Após isto, é abordado os conceitos e recursos de *Cloud Computing*, para alcançar o objetivo específico de criar e disponibilizar a solução do trabalho desenvolvido em nuvem.

1.2 Trabalhos Correlatos

O trabalho desenvolvido por (RADEMACHER; SORGALLA; SACHWEH, 2018b), visa demonstrar alguns desafios sobre o *Domain-Driven Design* em ambientes de microserviços, trazendo opções para lidar com eles através de desenvolvimento orientado a modelo, dando foco também em ferramentas de suporte para lidar com os desafios. Um dos desafios apresentados é definir o domínio da aplicação, pois normalmente são omitidas informações obrigatórias para deduzir um micro serviço e suas características, por exemplo, integrações entre sistemas, pois informações específicas são essenciais para a sua implementação.

Para lidar com esse desafio, deve-se escolher quais partes de domínios os times podem visualizar, tornando-as assim, acessível para eles. Também pode-se definir permissões para alterar os contextos delimitados, desde que tenham maturidade para mesclar mudanças e resolver dependências entre eles. Ainda sobre o exemplo da integração, para mitigar o problema é necessário criar interfaces de serviços modeladas para a porta de entrada (requisição) e porta de saída (resposta), resultando em uma abstração. Mantendo, assim, o contrato da integração e também desacoplado do coração do micro serviço.

Os autores demonstram em forma de tabela alguns padrões de *design* aplicados nos desafios propostos. A primeira linha da tabela, tem-se a Entidade do domínio que é única. Na segunda linha, relata-se sobre os objetos de Valores, que são normalmente imutáveis e precisam de uma identidade específica de domínio, podendo atuar na troca de informação de valor entre os contextos delimitados. As demais linhas da tabela, contêm os: Agregadores, Repositórios e Serviços. Por fim, na última linha relata sobre utilizar mais especificações ao invés de generalizações.

O trabalho desenvolvido por (LE; DANG; NGUYEN, 2020) aborda o desenvolvimento de software de forma iterativa ao modelos de domínio, que absorve todos os requisitos e que seja tecnicamente viável para implementação. Direcionando o foco, para o uso de uma linguagem de domínio específico interna, a uma linguagem de programação orientada a objetos com objetivo de construir o domínio. Assim o desenvolvimento de módulos de softwares utilizando DDD se torna generativo para o uso da linguagem.

O trabalho realizado por (MARSTON et al., 2011a) relata sobre a evolução da computação em nuvem que potencialmente é um grande avanço, mas para aproveitar dela é necessário uma visão clara de todos os pontos envolvidos tanto do cliente que utiliza, quanto do provedor que disponibiliza. E com tantos estudos e pesquisas acontecendo sobre essa tecnologia, é uma necessidade analisar vantagens e desvantagens da computação em nuvem. Neste estudo realizado, é citado vários problemas que podem afetar ambas partes interessadas da computação em nuvem e também, sobre recomendações para os profissionais que irão gerenciar toda essa infraestrutura.

O trabalho desenvolvido por (XU, 2012) visa mostrar como a computação em nuvem está mudando a forma com que empresas conduzem seus negócios, pois os componentes da infraestrutura neste formato são escaláveis e virtualizados sendo fornecidos como um serviço *web*. Com isso, são discutidos assuntos em relação aos clientes finais que utilizam a plataforma e até mesmo os provedores. Foram divididos em dois tipos de abordagens de computação em nuvem, a manufatura (com adoção de tecnologias de computação em nuvem) e a fabricação (como tecnologia da informação), disponibilizando plataformas e serviços sobre demandas com flexibilidade e sofisticação de soluções.

1.3 Organização do Trabalho

O trabalho foi dividido nos seguintes capítulos, introdução, trabalhos correlatos, conceitos adotados, desenvolvimento e conclusão. No capítulo de introdução, foi abordado sobre os temas, *Domain-Driven Design* e *Cloud Computing*, explorando a motivação de uso dos seus conceitos e práticas. O capítulo de de trabalhos correlatos, é relatado sobre trabalhos e artigos relacionados aos temas deste trabalho. Em seguida, os conceitos adotados, discorre sobre o que são os conceitos do *DDD*, pilares e suas aplicações. No seguinte, é falado sobre *Cloud Computing*, explorando o conceito e usabilidade. No desenvolvimento, são propostos casos de uso, para aplicar os conceitos do *Domain-Driven Design* e *Cloud Computing*. E por fim a conclusão, abordando desde a introdução até o final do desenvolvimento o que tira-se de conclusão sobre o trabalho.

2 Domain-Driven Design

2.1 Definição

O *DDD* relatado por [ERIC EVANS \(2016\)](#) define como um conjunto de princípios com foco em domínio, exploração de modelos de formas criativas, definir e falar a linguagem Ubíqua, baseado no contexto delimitado. Portanto, este trabalho irá abordar as definições que compõe o *DDD*, iniciando pela definição de domínio, que é um universo de conhecimento, atividade ou influência, ou seja, é o *core* do negócio, a razão para aquele software existir. O modelo que tem várias visões, que demonstra perspectivas de um determinado domínio, sem compreender a razão daquele software nascer, não é possível entender e modelar o sistema afim de ser mantido de médio e longo prazo. A linguagem ubíqua deve ser estruturada em conjunto dos especialistas do domínio, os membros que estão envolvidos no contexto delimitado e a equipe de desenvolvedores, com o objetivo de todos saírem falando a mesma linguagem referente ao contexto delimitado. O contexto é o significado em que aquele domínio está inserido, ou seja, o que é falado no modelo só é conhecido no seu domínio. E, por fim, o mais enfatizado contexto delimitado, é quem define a responsabilidade de cada subsistema, uma descrição de um limite, ou seja, com o domínio definido e a modelagem feita, consegue-se delimitar esses contextos. ([RADEMACHER; SORGALLA; SACHWEH, 2018a](#))

2.2 Pilares

Antes de começar a falar sobre os pilares do *DDD*, algumas características de modelos de domínio precisam ser analisadas. Esses modelos não podem depender de qualquer detalhe de implementação, não devem estar acoplados com a infraestrutura, devem ser capazes de serem reutilizados, devem ter uma fachada claramente separada, possibilitando manutenção e versionamentos, devem também estar concentrados em domínio de negócio específico, isto é, alinhados com o modelo de negócios, processos e estratégias. ([PENCHIKALA, 2022](#))([RADEMACHER; SORGALLA; SACHWEH, 2018a](#))

2.2.1 Linguagem Ubíqua

A linguagem Ubíqua faz parte dos pilares por ter um papel importante na aplicação do conceito. É necessário reunir os membros interessados e conhecedores do domínio de negócio, o time de desenvolvimento e pessoas envolvidas naquela iniciativa, para alinhar os termos e definições de negócio e técnica, convergindo a uma só linguagem a ser falada.

Visão Geral da Linguagem de Padrões

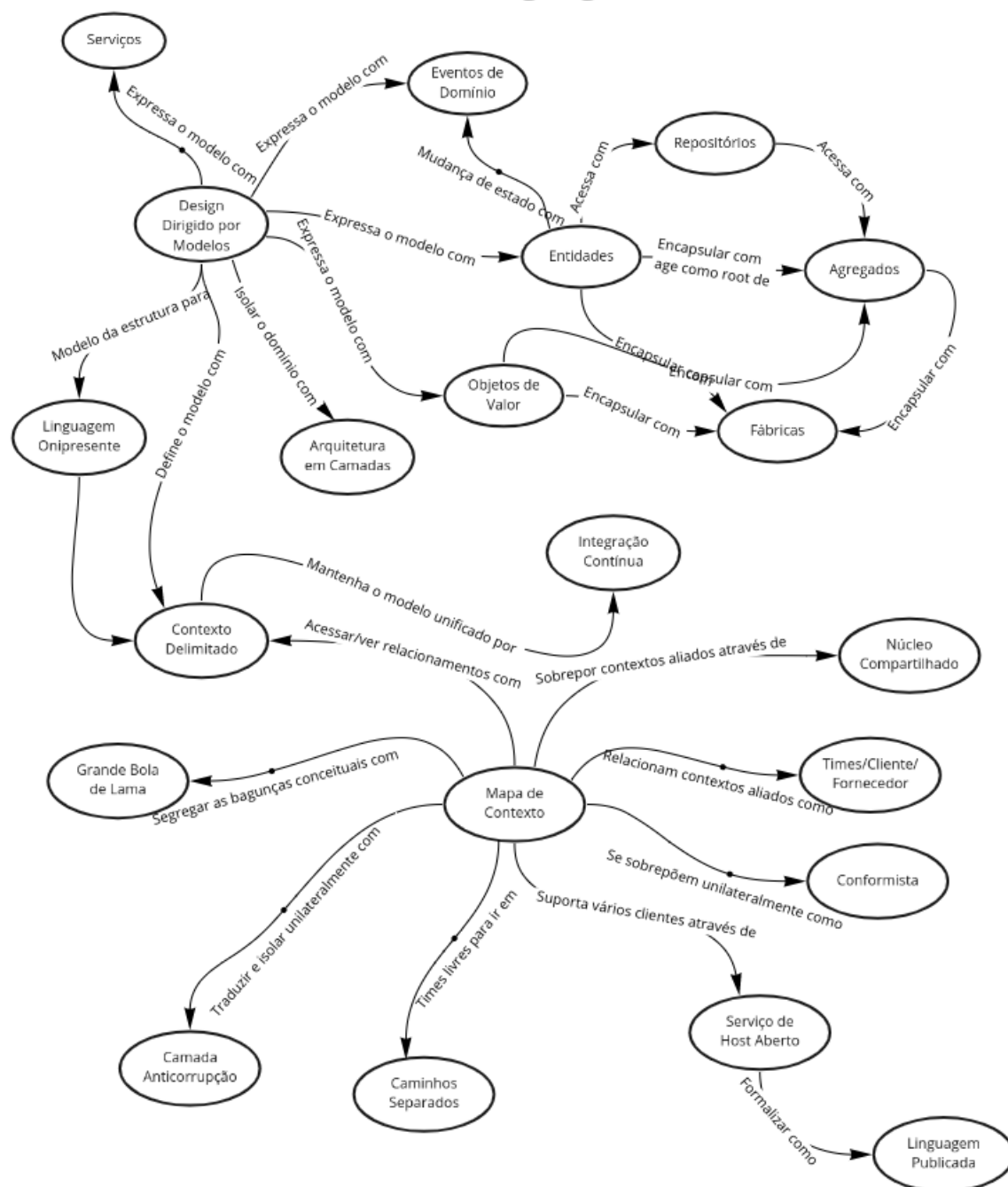


Figura 1 – Visão Geral da Linguagem de Padrões (ERIC EVANS, 2016)

Se o termo no contexto, por exemplo for "usuário", então a definição de usuário deve ser a mesma para o especialista do negócio naquele contexto e o desenvolvedor que irá atuar no mesmo contexto.([ERIC EVANS, 2016](#))

2.2.2 Contexto Delimitado

O *bounded context* é responsável por lidar com o limite entre os contextos da aplicação, definindo barreiras entre eles, pois cada um tem suas responsabilidades bem definidas. Também cada parte poderá ter a sua linguagem ubíqua específica. Por exemplo, quando pensamos em "cliente" no negócio de prestação de serviço é diferente do negócio de integração, veja que o cliente para uma prestação de serviço significa o cliente final, quem irá receber e pagar pelo serviço; já no conjunto de integração é o qual componente sistêmico capaz de consumir ou enviar algo para manter a comunicação entre os sistemas. Analisar as atividades que o especialista de negócio levantou como requisito, auxilia no processo de segregação de contexto. Quando o foco é ter casos de uso ou história de usuário, possibilita extrair possíveis separações de domínio de negócio.([ERIC EVANS, 2016](#))

2.2.3 Mapa de Contexto

Após a definição da linguagem comum à ser utilizada, os contextos e delimitações, segue-se então para o pilar do *context map*, em que serão estabelecidas as integrações entre os contextos gerando um mapa. Quando se define os domínios, separando por categorias como: principal, genérico e auxiliar, precisa-se mapear os contextos para estabelecer uma relação entre eles. O domínio principal, por exemplo, não pode sofrer alterações ao se relacionar com o domínio auxiliar ou genérico, então define-se uma relação de *upstream*, aquele que atua com ação sobre e *downstream*, aquele que sofre uma ação do outro. No contexto de cliente e fornecedor é onde o domínio principal exerce uma relação de *upstream* sobre o domínio genérico ou auxiliar. No caso de uma integração com *API* externa, que não seja de controle nosso, cria-se uma camada de anti corrupção, pois é preciso ser conformista com o contrato que foi estabelecido com a *API* externa e, se ela mudar, não é necessário alterar a implementação, mas sim, alterar o *layer* onde é integrado ([ERIC EVANS, 2016](#)).

2.2.4 Design Orientado por Modelos

Depois de todo o modelo e arquitetura já desenhados, pode-se partir para o *Building Blocks of a Model-Driven Design*, ou seja, Blocos de Construção de um Design Orientado a Modelos. Como citado por [EVANS \(2015\)](#), os blocos de construção de um design orientado por modelos são padrões que lançam práticas recomendadas de design orientado

a objetos a fim de instruir decisões para deixar mais claro o modelo e a implementação alinhados entre si, complementando e reforçando a eficácia do outro.

2.2.5 Arquitetura em Camadas

A *Layered Architecture* é um padrão que consiste em segregar a arquitetura de um software em camadas. Considerando uma arquitetura comercial, pode-se separar em quatro partes: infraestrutura, aplicação, usuário e domínio. A camada de infraestrutura consiste em uma abstração que auxilia todas as outras camadas, fornecendo implementação de comunicação, persistência, interface, entre outras. Ela tem o objetivo de ser uma biblioteca de suporte para as demais camadas. Já a interface do usuário, é aquela visão inteligente com as informações e interações com o usuário, sendo capaz de exibir e operar comandos. A camada de aplicação é quem orquestra as atividades da mesma, não mantendo estado de objetos de negócio, sem as lógicas do fluxo, mas pode armazenar o estado de uma atividade em progresso da aplicação. E a camada de domínio tem dados sobre o domínio do negócio. Os estados dos objetos de negócio são mantidos e para persistir os objetos é invocada a camada de infraestrutura. O objetivo é o isolamento, construindo um design em que não tenha acoplamento e que tenha coesão (PENCHIKALA, 2022)(RA-DEMACHER; SORGALLA; SACHWEH, 2018a).

2.2.6 Entidades e Agregadores

Entidades são definidas para representar objetos do domínio, criando uma identidade passando por ciclo de vida, mesmo que os seus atributos possam mudar. Deve ser capaz de distinguir cada um indiferente da sua forma ou histórico. Porém, os objetos contido como atributos, também são entidades, ou seja, são chamados de agregadores. Eles são entidades do mesmo contexto que utilizam uma das outras. É um aglomerado de objetos de domínio que podem ser tratados como uma única unidade. Portanto, deve se escolher quem é a raiz, cujo o nome dado é *aggregate root*(agregador pai) . As entidades possuem os seus atributos, métodos e validadores para garantir suas consistência, mas isso não significa que elas deverão conhecer o banco de dados, pelo contrário. Sendo transparente a camada de dados é puramente o objeto. (FOWLER, 2013)

2.2.7 Objetos de Valor

Diferente das entidades, o objeto de valor tem como objetivo representar algo que não tem identidade. Ao criar uma classe onde os atributos expressão igualdade no valor de suas propriedades são definidos como objetos de valor. Possibilitando ter uma coleção de atributos instanciados no construtor, eles são de origem imutável, ou seja, quando utilizado não podem sofrer efeitos colaterais. Ao contrário dos tipos primitivos, os objetos tem a sua tipagem forte. Portanto quando existe importância nos atributos de um elemento do

modelo e o mesmo expressa o significado, fazendo sentido em suas funcionalidades, assim é caracterizado como *Value Objects*. (FOWLER, 2013)(ERIC EVANS, 2016)

2.2.8 Repositórios

Quando é aplicado o conceito de arquitetura em camada, o resultado gerado faz com que o acoplamento diminua consideravelmente. Pode-se então, surgir necessidade de acessar camadas no sistema, por exemplo, banco de dados. Por isso surge o conceito de repositório, que é oferecido métodos para operar em objetos no banco de dados, sendo capaz de acessar direto a camada de dados consultando diretamente serviços externos. Ao utiliza-lo, o domínio começa a segregar a lógica de negócio de recursos externo, resultando na delegação de todos acesso e operações relacionados a banco de dados, através de repositórios.(FOWLER, 2013) (RADEMACHER; SORGALLA; SACHWEH, 2018a)

2.2.9 Serviços

Os serviços de domínio são responsáveis por realizar toda a lógica do negócio definida pelo especialista do domínio e para isso utiliza as agregações, entidades, repositórios como interface de acesso aos dados para conseguir aplicar a sua lógica do fluxo, também pode utilizar recursos de camadas de infraestrutura. Portanto quando a responsabilidade de Entidade ou Objetos de Valor não for capaz de resolver a lógica do negócio, é criada uma interface chamada Serviço com as operações declaradas como contrato para suprir essa necessidade.(ERIC EVANS, 2016)

3 Cloud Computing

3.1 Definição

O conceito de computação em nuvem, aborda a transição de informações e dados armazenados em lugar físico para ser disponibilizados em nuvem, através de grandes *Data Centers*. O principal objetivo é ofertar serviços de computação sob demanda para atender todas as necessidades de recursos de infraestrutura. O Instituto Nacional de Padrões e Tecnologia *NIST* define a computação em nuvem como: (VOUK, 2008) (XU, 2012)

“um modelo para permitir acesso de rede onipresente , conveniente e sob demanda a um conjunto compartilhado de recursos de computação configuráveis (por exemplo , redes , servidores , armazenamento , aplicativos e serviços) que podem ser rapidamente provisionados e liberados com o mínimo esforço de gerenciamento ou interação do provedor de serviços.”

3.2 Computação em Nuvem Como Serviço

Os principais tipos de serviços da computação em nuvem são: plataformas, infraestruturas e softwares. Que são disponibilizados com segurança através de recursos abrangentes para cumprir os requisitos mais rigorosos. O modelo híbrido de infraestrutura, possibilita criar arquiteturas que estendem de *on-premises*, que são recursos disponibilizados em máquinas físicas, que utilizam a sua rede de internet, para a nuvem. O provedor garante controles sofisticados dos serviços, auditoria, credenciamentos de segurança abrangentes e escalabilidade. Sendo possível acessar a quantidade de recursos do que precisar e escalando-se conforme as necessidades em apenas alguns minutos. (MARSTON et al., 2011b)

3.3 Software como um serviço

O *SaaS* Software como um serviço oferece uma plataforma que reuni vários recursos comuns gerenciado pelo provedor sem a necessidade de preocupar como está sendo executado, mantido ou orquestrado. Por exemplo o *webmail* da provedora de nuvem AWS (2022a) no qual você pode enviar e receber e-mails sem precisar gerenciar recursos adicionais para o produto de e-mail ou manter os servidores e sistemas operacionais no qual o programa de e-mail está sendo executado.(MARSTON et al., 2011b)

3.4 Plataforma como um serviço

A *PaaS* Plataforma como um serviço disponibiliza todos os recursos necessários para o ciclo de vida de um software no seu desenvolvimento, seja eles armazenamento de dados, hospedagem de aplicações, testes e todo o gerenciamento para que se preocupe apenas com a implantação dos serviços. Isolando o esforço não mais para a escalabilidade, aquisição de recursos ou planejamento de capacidade. (DIKAIKOS et al., 2009)

3.5 Infraestrutura como um serviço

A *IaaS* Infraestrutura como um serviço oferece recursos e acessos para possibilitar analisar os armazenamentos de dados e rede, capacidades de serviços executados flexibilizando o controle de gerenciamento sobre os recursos provisionados. Também promove o pagamento na utilização sob demanda, sem preocupar com a evolução e inovação da tecnologia. Permitindo que administradores operam na plataforma com acesso a tudo que está sendo provido. E sendo atendidos os requisitos solicitados da grande parte de clientes como *On-demand*, auto-sustentável ou auto-reparável, *multi-tenant* e segregação de clientes. (MARSTON et al., 2011b)

3.6 Provedor de nuvem Amazon Web Services (AWS)

Dentre os provedores de nuvem encontrados no mercado como: *Microsoft Azure*, *Google Cloud Platform (GCP)*, *Oracle Cloud*, *IBM Cloud*, *DigitalOcean* entre outros, irá ser abordado e utilizado no desenvolvimento especificamente a *AWS*. A escolha deste provedor, é exclusivamente por familiaridade. (AZURE, 2023) (GOOGLE, 2023) (ORACLE, 2023) (IBM, 2023) (OCEAN, 2023)

A Amazon Web Services (AWS) é a plataforma de nuvem mais adotada e mais abrangente do mundo, oferecendo mais de 200 serviços completos de datacenters em todo o mundo. Milhões de clientes, incluindo as startups de crescimento mais rápido, grandes empresas e os maiores órgãos governamentais, estão usando a AWS para reduzir seus custos, ficarem mais ágeis e inovarem mais rapidamente. (AWS, 2022b)

Portanto para esse trabalho optou-se por utilizar os recursos do provedor de nuvem *AWS*.

3.6.1 AWS Elastic Beanstalk

Um dos recursos que auxilia a implantar um software na nuvem, sem ter que gerenciar toda a infra estrutura é o *Elastic Beanstalk*. Com ele é possível através de uma

interface subir um *software* e deixar disponível para ser acessado. Pois esse *software* vai ser automaticamente gerenciado pelo recurso, possuindo um balanceamento de carga, escalabilidade, provisionamento de capacidade e monitoramento da integridade da aplicação. Portanto ele é um serviço de orquestração para a implantação de aplicativos que orquestram serviços como: *Amazon EC2*, *Amazon DynamoDB*, *Elastic Load Balancing*, *Amazon S3*, *Amazon Simple Notification Service*, *Amazon Simple Queue Service*, *Amazon RDS* entre outros.

Com a funcionalidade de orquestração, o *Elastic BeanStalk* segue um fluxo para disponibilizar o recurso na nuvem. As etapas do fluxo de trabalho dele são: criação da aplicação, nessa etapa é criada uma aplicação, após isso, é fazer o envio dessa versão para ser carregada, depois lançar em seu ambiente uma versão e para finalizar é feito todo o gerenciamento da aplicação. A figura abaixo representa o fluxo de trabalho do *Elastic Beanstalk*.(BENSTALK, 2022)

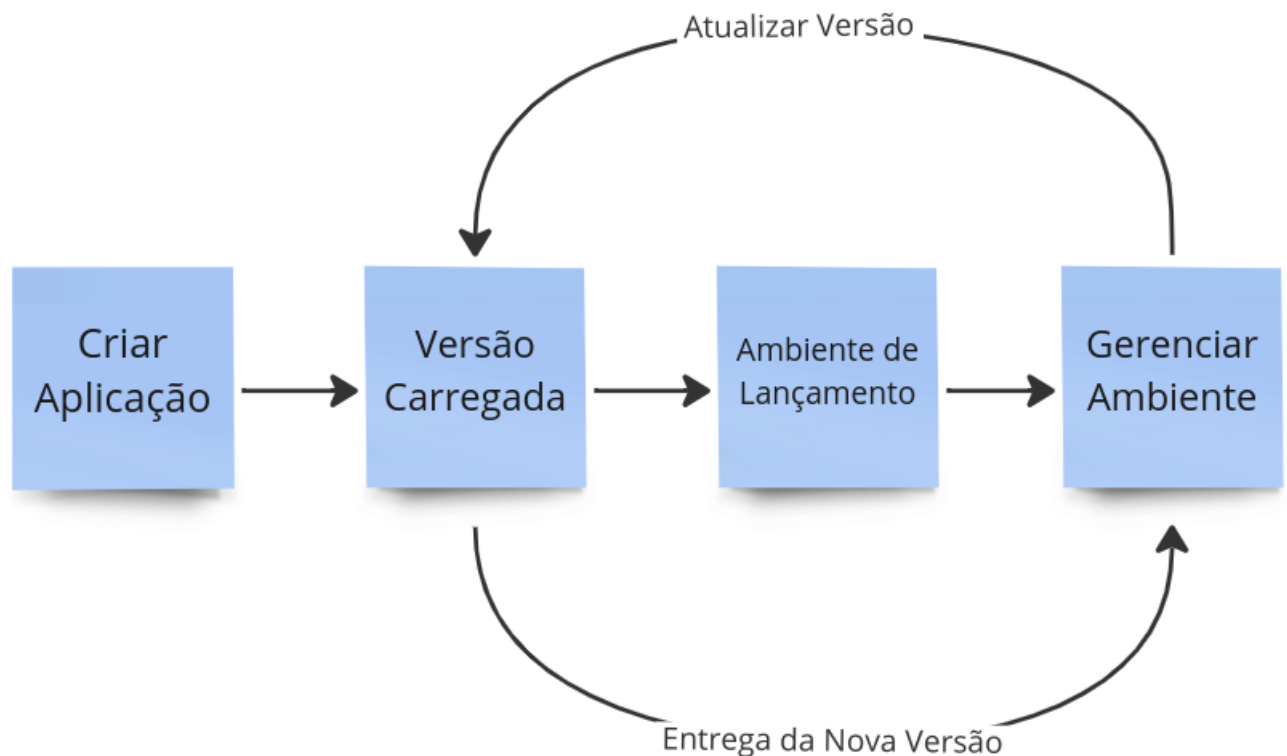


Figura 2 – Fluxo de trabalho do Elastic Beanstalk. (AWS-BENSTALK, 2022)

3.6.2 Amazon Elastic Compute Cloud

O *Amazon EC2* é um serviço web que permite que os usuários aluguem computadores virtuais para executarem suas aplicações.

Ele fornece uma infraestrutura com capacidade computacional escalável sob demanda, segura e redimensionável, com mais de 500 instâncias e opções de processador

que permite executar e criar virtualmente qualquer aplicação.

Disponibiliza funcionalidades como armazenamento, sistema operacional e redes para uma computação segura, com performance otimizada para as aplicações. (AWS-EC2, 2022)

3.6.3 Amazon DynamoDB

Para uma solução de banco de dados não relacional *NoSQL* orientada a chave e valor, a Amazon oferece um serviço rápido e flexível com performance abaixo de 10 milissegundos em qualquer escala.

A imagem abaixo mostra os principais recursos desse banco de dados, ao lado esquerdo na primeira seção, com o título *Configure Key Feature* mostra que é uma plataforma sem servidor e totalmente gerenciado, que disponibiliza backups contínuos, replicação, segurança integrada, multi regiões, com cache na memória e ferramentas de exportação de dados.

(AWS-DYNAMODB, 2022)

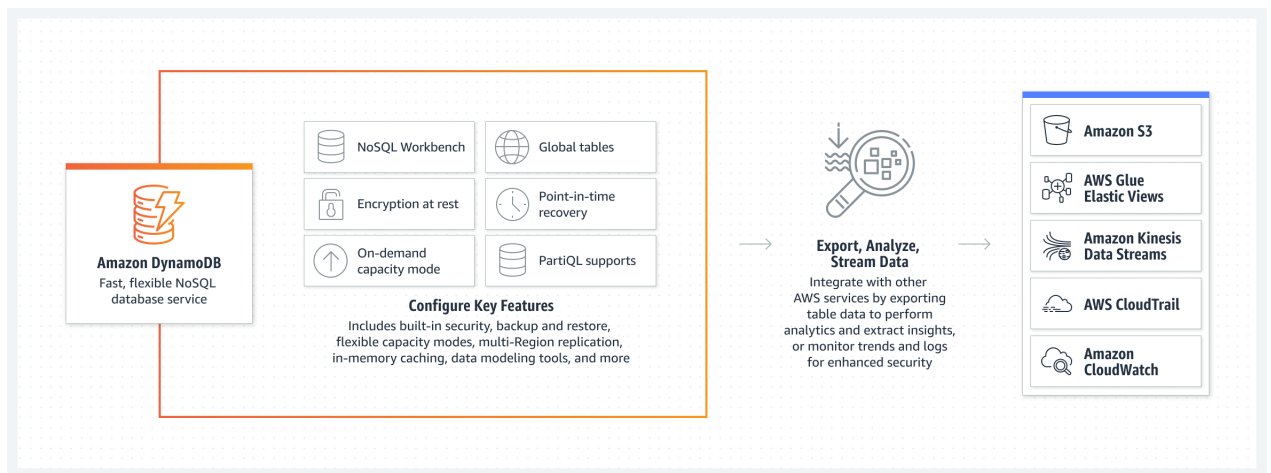


Figura 3 – Características do DynamoDB. (AWS-DYNAMODB, 2022)

Ainda nessa seção, tem-se seis caixas que estão escritas com mais funcionalidades dele como *NoSQL Workbench* sendo uma plataforma não relacional, criptografia dos dados em repouso opera sob demanda, trabalha com tabelas globais proporcionando o acesso delas em várias regiões, recuperação dos dados através de um ponto definido, suportes do *PartiQL*, uma linguagem que garante acesso de consultas compatíveis com *SQL* em vários armazenamentos de dados. na segunda seção, há uma seta à esquerda que significa a chegada de dados do *DynamoDB* e um seta à direita que significa a saída de dados para os produtos da *AWS* aos quais o banco de dados se integra.

E na última seção do lado direito, tem-se cinco caixas que representam quais os recursos ele se integra por padrão.

3.6.4 Elastic Load Balancing

Uma solução para balancear a carga de de tráfego auxiliando na escalabilidade da aplicação, a amazon oferece um serviço que distribui automaticamente a carga de aplicações de entrada entre vários destinos em uma ou mais zonas de disponibilidades.

Na figura abaixo, ao lado esquerdo dentro de uma caixa titulada como *Source*, mostra os protocolos que ele dão suporte.

([AWS-ELB, 2022](#))

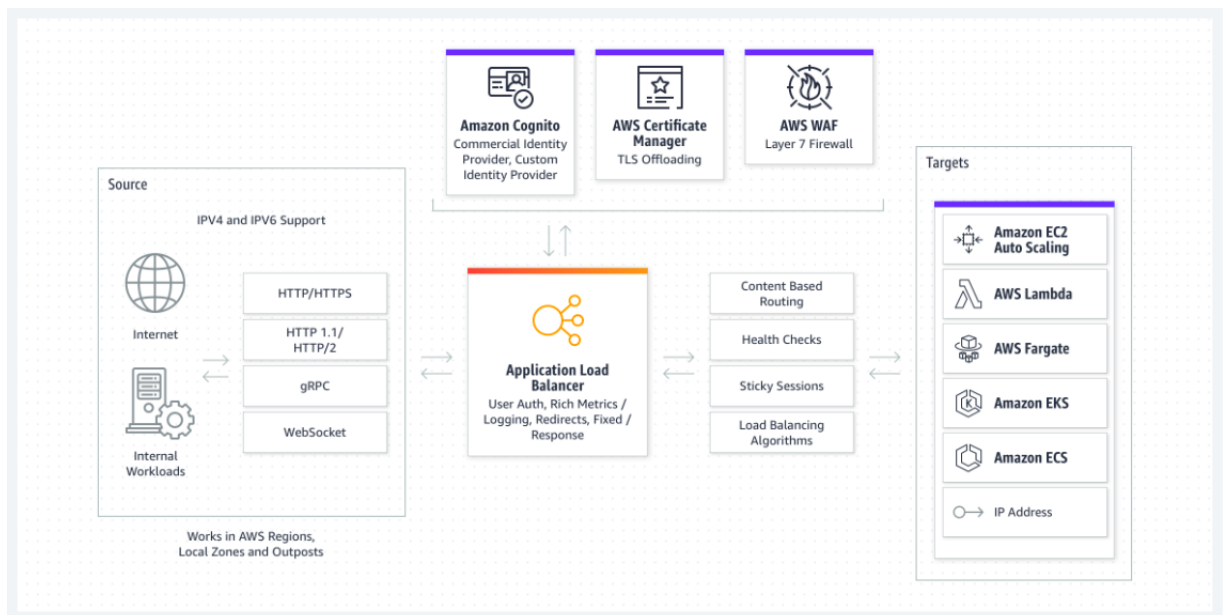


Figura 4 – Elastic Load Balancing.

Logo no centro da imagem, mostra o balanceamento saindo ou chegando dos recursos *Amazon Cognito*, *AWS Certificate Manager* e *AWS WAF*, são componentes com responsabilidades de indentidade de provedor, gerenciamento de certificados e *firewall* da AWS.

Por fim, ao lado direito mostra seis caixas com componentes de aplicações do provedor, onde é possível atuar com o balanceador de carga.

3.6.5 Amazon S3

O *Simple Storage Service* da AWS é um serviço que oferece armazenamento de objetos que possibilita guardar e recuperar qualquer volume de qualquer lugar. Também

altamente escalável com segurança e podendo ser gerenciado com certa facilidade.

A imagem abaixo mostra como mover dados para o *bucket*, ao lado esquerdo, tem-se as origens e diferentes tipos de dados como os dados analíticos, arquivos de log, dados de aplicações, vídeo e imagens e “backup e arquivamento.



Figura 5 – Amazon Simple Storage Service (Amazon S3). ([AWS-S3, 2022](#))

No centro da imagem, demonstra as funcionalidades dele, como: ao criar um *bucket* pode-se especificar a região, controlar os acessos e opções de gerenciamento, carregar qualquer quantidade de dados.

Na seção ao lado, tem-se seis caixas que também são seus recursos, tais como: otimização dos custos com classes de armazenamento, replicação de dados para qualquer região, acesso *on-premises* ou via *VPC*, proteção de dados e visibilidades de armazenamentos.

A última seção ao lado direito, é sobre a análise dos dados, ou seja, pode-se usar recursos do provedor para obter métricas e *insights* sobre os dados e arquivos armazenados. ([AWS-S3, 2022](#))

3.6.6 Amazon Simple Notification Service

A *AWS* oferece também um serviço de notificação que possibilita notificar SMS, e-mail, push móveis totalmente gerenciadas. Fornecendo tópicos para que aplicações possa enviar mensagens de alta taxa de transferência para outros sistemas, em uma arquitetura distribuída facilita a comunicação entre micro serviços possibilitando ser notificado através de envio de mensagens, consegue-se uma visão macro do como funciona com a imagem abaixo.

([AWS-SNS, 2022](#))



Figura 6 – Amazon Simple Notification Service. (AWS-SNS, 2022)

Nessa imagem, ao lado direito tem-se um publicador, ele pode ser um micro serviço ou outros recursos do provedor de nuvem.

Ao centro, é mostrado que ao receber uma mensagem, a mesma é enviada um tópico, onde pode ser feito filtros ou regras para a replicação da mensagem e até mesmo enviar para um tópico novo, aquela ao qual não conseguiu-se processar.

E na última seção ao lado direito, tem-se os componentes que podem se inscrever para ser notificados de mensagens.

3.6.7 Amazon Simple Queue Service

Outro recurso que o provedor de nuvem disponibiliza são filas de mensagens gerenciado que auxilia no desacoplamento e a escalabilidade de micro serviços, sistema distribuídos com aplicações sem servidores. Podendo utilizar dois tipos de filas de mensagens, com *throughput* máximo, com o melhor trabalho de classificação e entrega pelo menos uma vez.

As filas do *SQS* garante também que as mensagens serão processadas exatamente uma única vez, em ordem que foram enviadas.

Na figura abaixo, demonstra o seu fluxo de trabalho. Ao lado esquerdo, tem-se um produtor de mensagem que faz o seu disparo para o *SQS*.

SQS.(AWS-SQS, 2022)

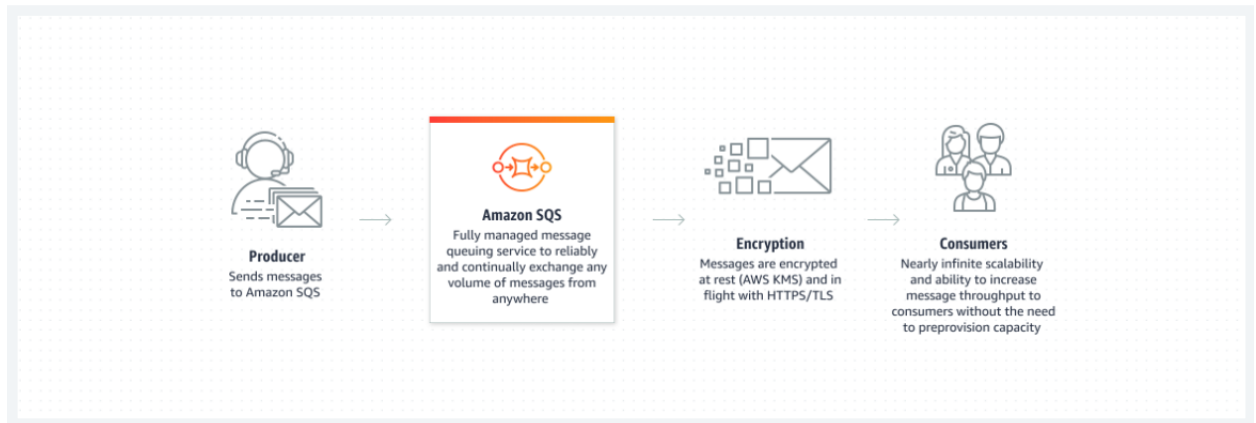


Figura 7 – Amazon Simple Queue Service. ([AWS-SQS, 2022](#))

Em seguida ao seu lado direito, é mostrado o componente em si. Ao seu lado, através do ícone de uma carta, é dito que pode-se criptografar a mensagem com o próprio recurso do provedor com o nome *AWS KMS*, o qual permite gerenciar e armazenar criptografias.

Ao final do lado direito, tem-se os consumidores dessa fila

3.6.8 Amazon RDS

O *Amazon Relational Database Service* é um recurso que possibilita a utilização de base de dados relacionais em nuvem.

São serviços gerenciados e de fácil configuração, manutenção e escalabilidade, com isso é ofertado algumas opções de de bases de dados como *SQL Server*, *Oracle*, *MySQL*, *PostgreSQL*, *MariaDB* entre outros.

A imagem a seguir, mostra os benefícios do recurso, separadas em três seções.

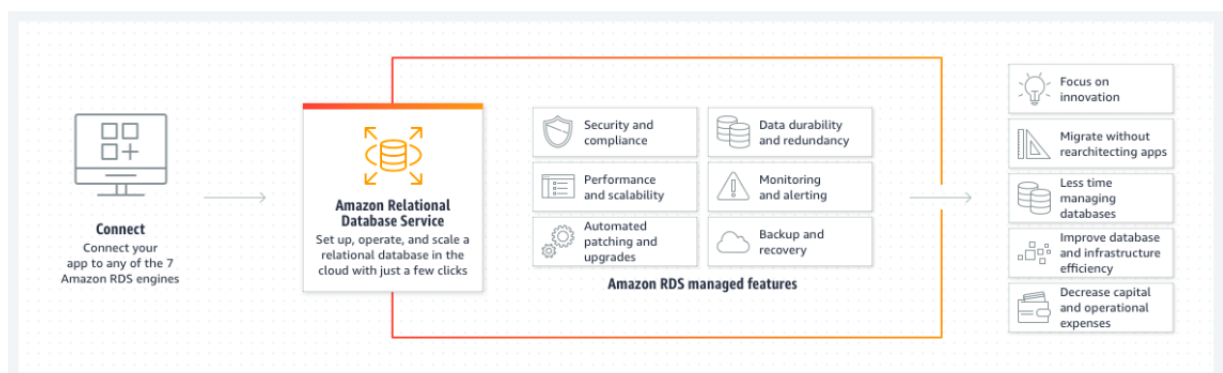


Figura 8 – Amazon Relational Database Service ([AWS-RDS, 2022](#))

Na primeira ao lado esquerdo, tem-se um ícone de um computador, significando a conexão de algum aplicativo através de um mecanismo do *AWS RDS*.

A seguir, estão as funcionalidades que o recurso disponibiliza, como: operações e escalabilidade de um banco de dados relacional na nuvem, segurança e conformidade, desempenho e escalabilidade, atualizações e aplicação de pacotes automatizados, durabilidade e redundância dos dados, monitoramento e alerta e backup e recuperação.

Por fim, na ultima seção ao lado direito, mostra de fato os seus benefícios, listados em cinco caixas, descritas: foco na inovação, migrar sem redefinir a arquitetura de aplicações, diminuir o tempo de gerenciamento de bancos de dados, melhorar a eficiência do banco de dados e da infraestrutura e reduzir as despesas de capital e operacionais. ([AWS-RDS, 2022](#))

4 Desenvolvimento

4.1 Aplicação da modelagem utilizando conceito do DDD

Para esse trabalho foi aplicado a modelagem utilizando o *DDD* para uma solução que tem como objetivo possibilitar cadastrar e consultar os dados de um cliente, atualizar esse cadastro exceto o documento anexado à esse cliente, consultar operações realizadas pelo administrador do sistema referente ao cliente e consultar a situação do mesmo. Portanto pode se começar a exploração da necessidade e os casos de uso do sistema para iniciar a modelagem.

4.1.1 Glossário

Uma das formas de utilizar o conceito de linguagem ubíqua é montar um glossário ou dicionário com as palavras chaves descritas nos casos de uso, com isso é possível extrair as palavras chaves que podem ter significados diferentes de acordo com o contexto e definir um significado comum a todos envolvidos no ecossistema do produto para que todos possam falar a mesma língua. Com base nos requisitos, foi construído o seguinte glossário:

- Cliente - Refere-se como Cliente, o código que o identifica dentro do sistema.
- Cadastro - São todos os dados pessoais referente ao cliente em nosso sistema.
- Endereço - Representa o endereço da residência ou trabalho de um Cliente.
- Documento - É o número único de cadastro de pessoa física em nosso País que identifica o Cliente.
- Observação - Detalhes daquele Cliente cadastrado em nosso sistema.
- Operação - É toda a ação feita pelo Administrador que tem acesso a plataforma em nosso sistema.
- Documento Anexado - É o documento anexado no cadastro.

4.1.2 Contexto Delimitado em Caso de Uso

O próximo passo agora vai ser a delimitação do contexto com base em cada caso de uso listado abaixo:

- Disponibilizar todos os dados de um cadastro de **Cliente**.

- Possibilitar o cadastro do **Cliente**.
- Possibilitar atualizar os dados de cadastro do **Cliente** exceto o documento anexado.
- Consultar **Situação** do Cliente no mercado.
- Consultar as **Operações** realizadas pelo Administrador do sistema sobre o **Cliente**.
- Ao realizar busca do **Cliente** , trazer o documento anexado como link para acesso.

Percebe-se que as palavras chaves estão negrito, que define os contextos, pois são pontos focais caracterizando domínios da aplicação. Portanto é possível ter em mente já as entidades que representam o sistema, que são elas: *Cliente*, *Operação* e *Situação*. Categorizando como domínios, o Cliente é o principal pois sem ele o sistema não funcionaria, grande parte das funcionalidades são aplicadas em cima desse domínio. Seguindo para a *Operação*, nota-se que ela ajuda em todo o processo fazendo com que o domínio principal funcione e tem como característica genérico porque ele serve várias coisas e trabalha independentemente. Por fim a *Situação* é um domínio auxiliar, pois diferente de um domínio genérico ela auxilia os outros domínios executarem suas funções. Abaixo está a modelagem estratégica da delimitação de contextos:

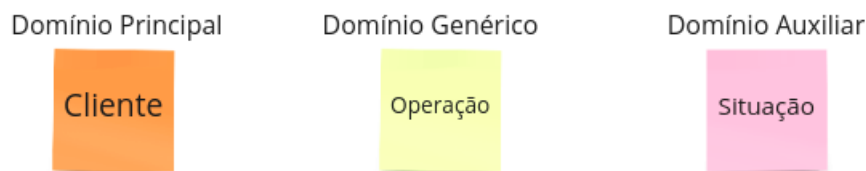


Figura 9 – Modelagem Estratégica

4.1.3 Mapa de Contextos

O mapeamento dos contextos delimitados (*Bounded Contexts*), estabelece relação entre domínios. O principal tem uma relação de *upstream* para o genérico, ou seja, o contexto de Cliente é produtor para a Operação que é consumidor, isso porque o domínio principal é prioridade em relação ao genérico, caso tenha alguma alteração, terá que ser do lado do domínio genérico. Na relação entre domínio auxiliar e o principal, é estabelecida sobre o Cliente ser produtor e a Situação consumidor, ou seja, neste caso não é conhecido o sistema externo que a *API* de consulta opera, então é criada uma camada anti-corrupção com o objetivo de isolar esse sistema externo do contexto. E com isso existe uma relação de conformismo, pois é possível adaptar a fachada do sistema, para evitar que seja acoplado, com o sistema externo caso ocorra mudanças na integração deles. E por

fim toda a arquitetura está em um núcleo compartilhado, ou seja, todos podem acessar esse processo e funcionalidade do sistema. Em seguida é mostrado visualmente o mapa de contexto.

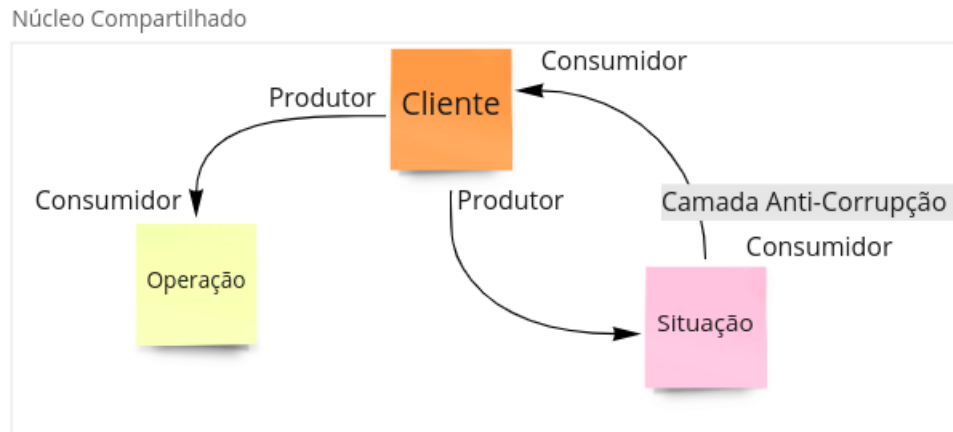


Figura 10 – Mapa de Contexto

4.1.4 Arquitetura Contextual

Neste momento é possível avançar para a arquitetura técnica da modelagem do sistema. Foi criado um serviço e um componente para atender as funcionalidades do *BackEnd*, todos eles escritos na linguagem de programação Kotlin, se comunicam através de integração via transferência de estado representacional *REST*, que é um conjunto de restrições de arquitetura. E a interface desenvolvida com *Angular* para disponibilizar o cadastro, busca e alteração. A estratégia de entrega desses sistemas na plataforma de nuvem é através do *Elastic BeanStalk* facilitando todo o gerenciamento das aplicações. Para o serviço cliente-api é utilizando um banco de dados não relacional *noSQL* chamado *DynamoDB*, também recurso do provedor de nuvem. E para atender o serviço de operacao-api é utilizado um banco de dados relacional *postgresql* que é executado através do O Amazon Relational Database Service, ou Amazon *RDS*, que é um serviço de banco de dados relacional distribuído da (AWS, 2022b). do provedor de nuvem. Logo abaixo é possível visualizar a integração entre os sistemas e componentes da arquitetura de uma maneira simples.

Ainda sobre as tecnologias que vão ser abordada no desenvolvimento, a linguagem de programação que irá ser utilizada é o Kotlin, (LANG, 2023). A biblioteca que irá auxiliar disponibilizando recursos para integrações, servidor da aplicação entre outras é o *Spring Framework* (SPRING, 2023). Para a interface irá ser utilizado o *Angular* (ANGULAR, 2023) que disponibiliza a interface e integração com os outros sistemas da arquitetura. Para versionar os códigos dos sistemas, irá ser utilizado como repositório e gerenciamento o *GitHub* (GITHUB, 2023). E para a criação da infraestrutura como código

será utilizado o *Terraform*([HASHICORP, 2023](#)).

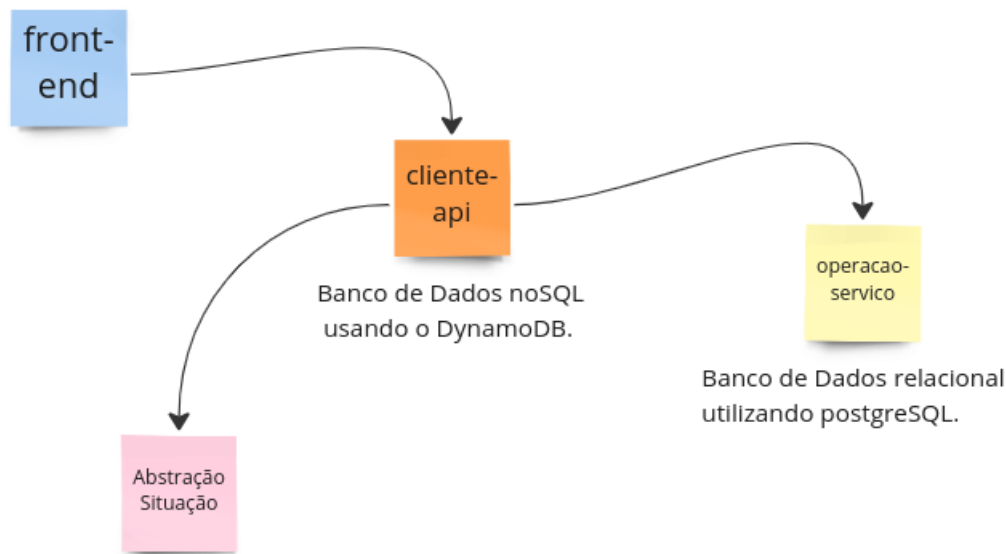
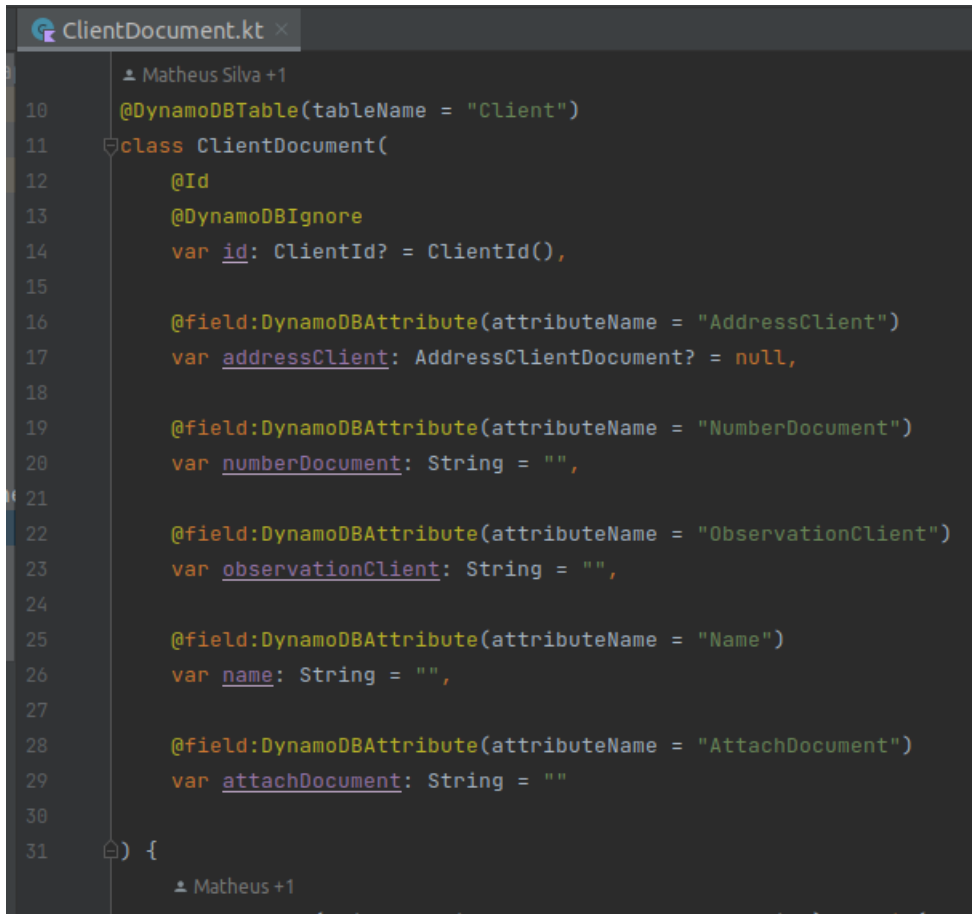


Figura 11 – Arquitetura Contextual

4.2 Implementando a aplicação de Cliente

Para disponibilizar as funcionalidades dos casos de uso relacionados ao cadastro de clientes, foi criado a *API* cliente-api, que é responsável por criar, atualizar e buscar um cliente, retornar as operações do administrador do sistema e a situação do cliente. A responsabilidade única dessa aplicação é gerenciar o seu domínio e fornecer dados relacionados a clientes. Para se atender o requisito de busca de operações, foi desenvolvida uma integração com a *API* de operações. Pois mesmo que os dados sejam relacionados, as responsabilidades de domínios são diferentes. Resultando em não se armazenar esses dados de operações no contexto de Cliente.

A entidade de Cliente, com a responsabilidade de representar os dados que serão armazenados no banco de dados, foi construída através de uma classe que representa o documento que será armazenado. A imagem a seguir, mostra seus atributos.

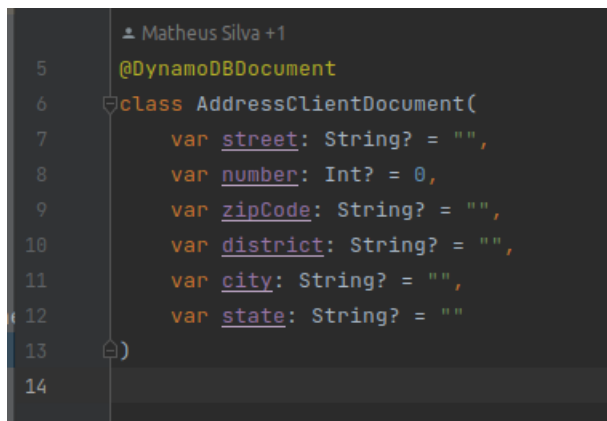
A screenshot of an IDE showing the Kotlin code for a class named ClientDocument. The code is annotated with AWS DynamoDB annotations. The class has a primary key 'id' and several other attributes: addressClient, numberDocument, observationClient, name, and attachDocument. The code is as follows:

```
10 @DynamoDBTable(tableName = "Client")
11 class ClientDocument(
12     @Id
13     @DynamoDBIgnore
14     var id: ClientId? = ClientId(),
15
16     @field:DynamoDBAttribute(attributeName = "AddressClient")
17     var addressClient: AddressClientDocument? = null,
18
19     @field:DynamoDBAttribute(attributeName = "NumberDocument")
20     var numberDocument: String = "",
21
22     @field:DynamoDBAttribute(attributeName = "ObservationClient")
23     var observationClient: String = "",
24
25     @field:DynamoDBAttribute(attributeName = "Name")
26     var name: String = "",
27
28     @field:DynamoDBAttribute(attributeName = "AttachDocument")
29     var attachDocument: String = ""
30
31 ) {
```

Figura 12 – Entidade Cliente

A propriedade `id` da classe, representa a chave primária que será utilizada pelo *DynamoDB* com a finalidade de armazenar registros únicos dentro da tabela.

O endereço do Cliente, é mapeado dentro da classe `addressClient`, que representa os seguintes atributos:

A screenshot of an IDE showing the Kotlin code for a class named AddressClientDocument. The code is annotated with AWS DynamoDB annotations. The class has several attributes: street, number, zipCode, district, city, and state. The code is as follows:

```
5 @DynamoDBDocument
6 class AddressClientDocument(
7     var street: String? = "",
8     var number: Int? = 0,
9     var zipCode: String? = "",
10    var district: String? = "",
11    var city: String? = "",
12    var state: String? = ""
13 )
```

Figura 13 – Documento Endereço

Estes atributos, representam a informação de endereço do Cliente, respectivamente

por: rua, número, código postal, bairro, cidade e estado. Já o número de identificação do Cliente, é representado pela propriedade `numberDocument`, com o objetivo de ser armazenado apenas o que identifica esse documento. O atributo `observationClient`, representa observações sobre o cadastro do cliente, possibilitando armazenar informações sobre ele. Seguindo a ordem das propriedades da classe, o atributo `name`, representa o nome do cliente a ser armazenado.

O último atributo da classe é `attachDocument`, que representa o endereço de acesso ao documento anexado no cadastro do cliente. Como parte do fluxo desenvolvido, ao anexar um documento, é feito o envio para o *s3*, um serviço que permite armazenar documentos de diferentes formatos. E recuperado o endereço para acesso do mesmo.

4.2.1 Arquitetura do Projeto

A arquitetura do sistema foi dividida em pacotes conforme a imagem abaixo:

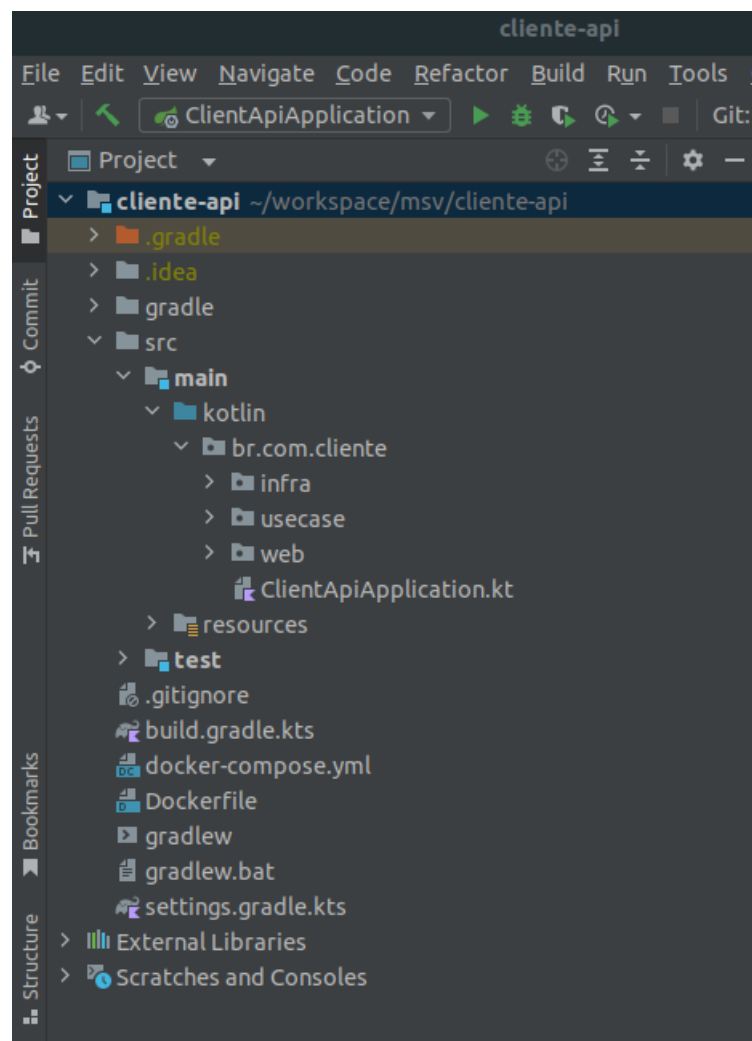


Figura 14 – API de Cliente

O pacote de infra, é responsável por guardar classes, arquivos, tipos enumerados e interfaces que estão relacionados diretamente a integrações que a aplicação utiliza. A imagem abaixo mostra os pacotes que são utilizados.

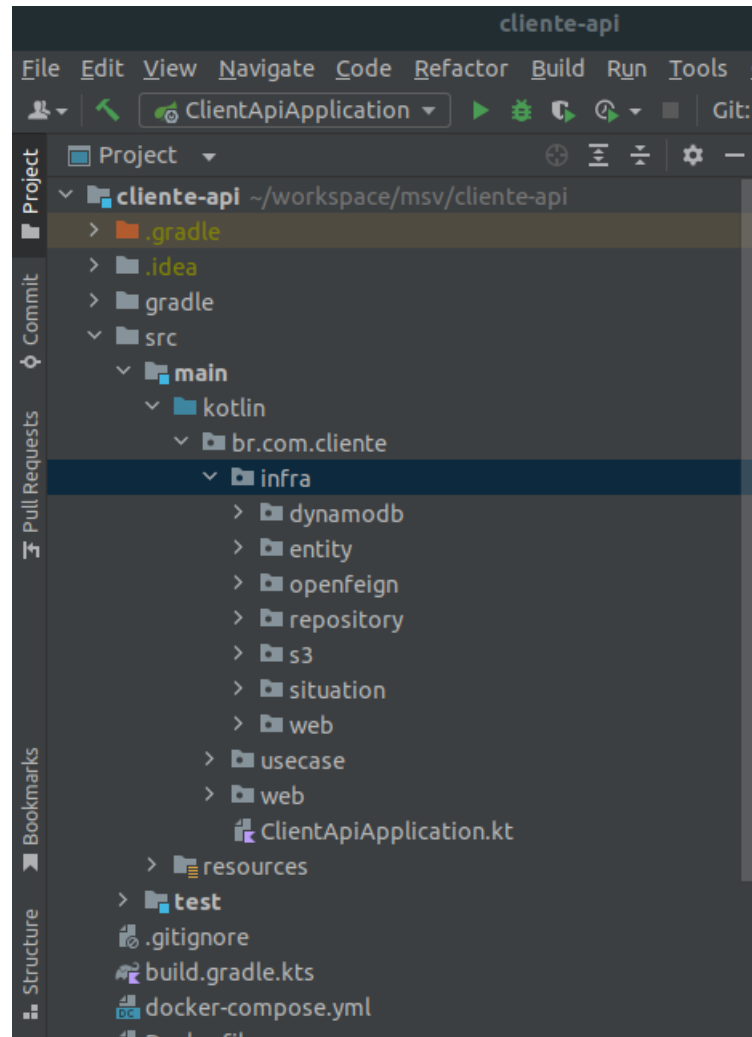


Figura 15 – Pacote Infraestrutura cliente-api.

Dentro do *dynamodb*, está a classe responsável por prover as configurações de conexão e acesso ao banco de dados não relacional.

Na pasta *entity* estão as entidades do domínio, para esse banco de dados temos classes que representam documentos que modelam a tabela do *Dynamodb*.

Logo em seguida no pacote *openfeign*, têm a interface para a integração com a *API* de operações junto com o pacote que armazena a classe que represente a requisição feita.

Já no *repository*, contem a interface com a responsabilidade de persistir e buscar registros na base de dados. As classes de configurações e da integração com o *Bucket S3* estão no respectivo pacote.

A fachada que fornece a consulta de busca de situação do cliente, está no pacote

situation.

A configuração para habilitar a integração da interface está em classe dentro do pacote *WEB*.

No pacote de *usecase*, estão os casos de uso implementados pela aplicação, que são responsáveis pelos fluxos do sistema. Contendo as seguintes classes:

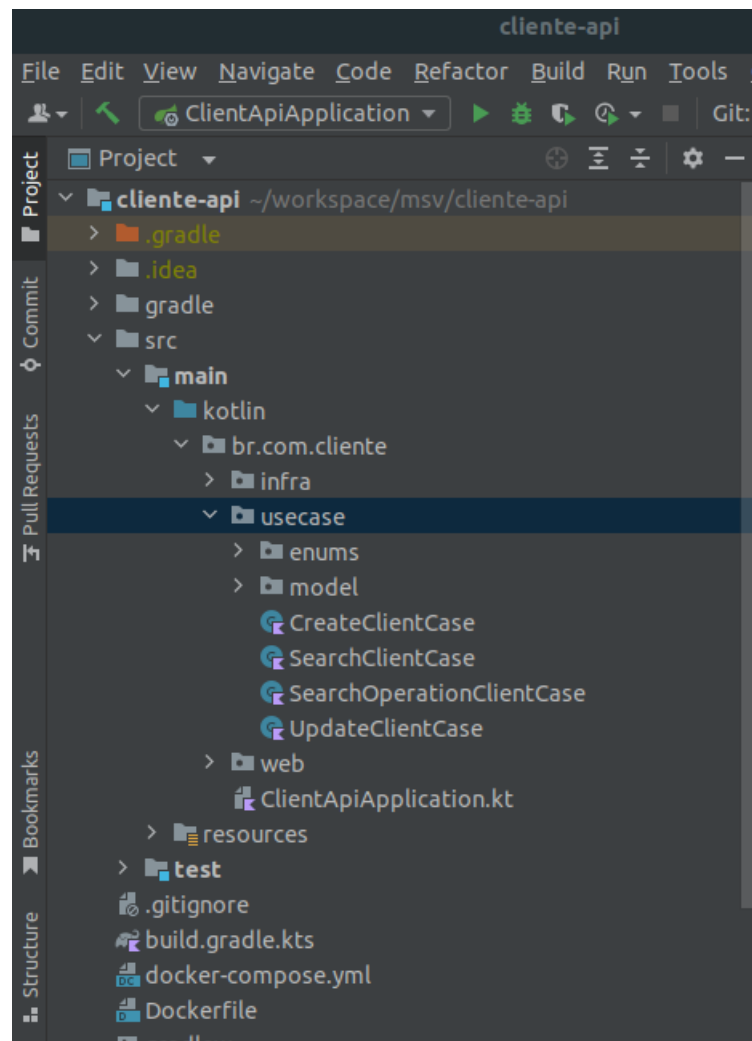


Figura 16 – Pacote Casos de Uso cliente-api.

Os dois sub pacotes: *enums* e *model*, são responsáveis por representar a enumeração da situação e as classes que definem o modelo que é trabalhado os domínios. Restando as classes responsáveis pela lógica de negócio da criação, buscas e atualização.

Por fim, no pacote *web*, são mapeadas as classes que representam requisições recebidas e retornadas do sistema. E a classe que representa o recurso que contém os métodos de integrações *REST*.

As propriedades armazenadas para a execução da aplicação, estão no pacote *resources* no arquivo *application.yml*.

Nessa arquitetura foram aplicados os padrões de arquitetura em camadas, segregando a infraestrutura da aplicação e do domínio. Entidades e Agregadores que representam os objetos do domínio. Repositório que é formado por interface que auxilia para diminuir o acoplamento. E similar ao padrão de serviços do *DDD* foi dividido em casos de usos.

4.3 Construção do componente de Operações

Foi construído para atender os requisitos de criação e busca de operações realizadas pelo usuário administrador do sistema. Semelhante a *API* de clientes, a arquitetura segue em camadas conforme a imagem abaixo:

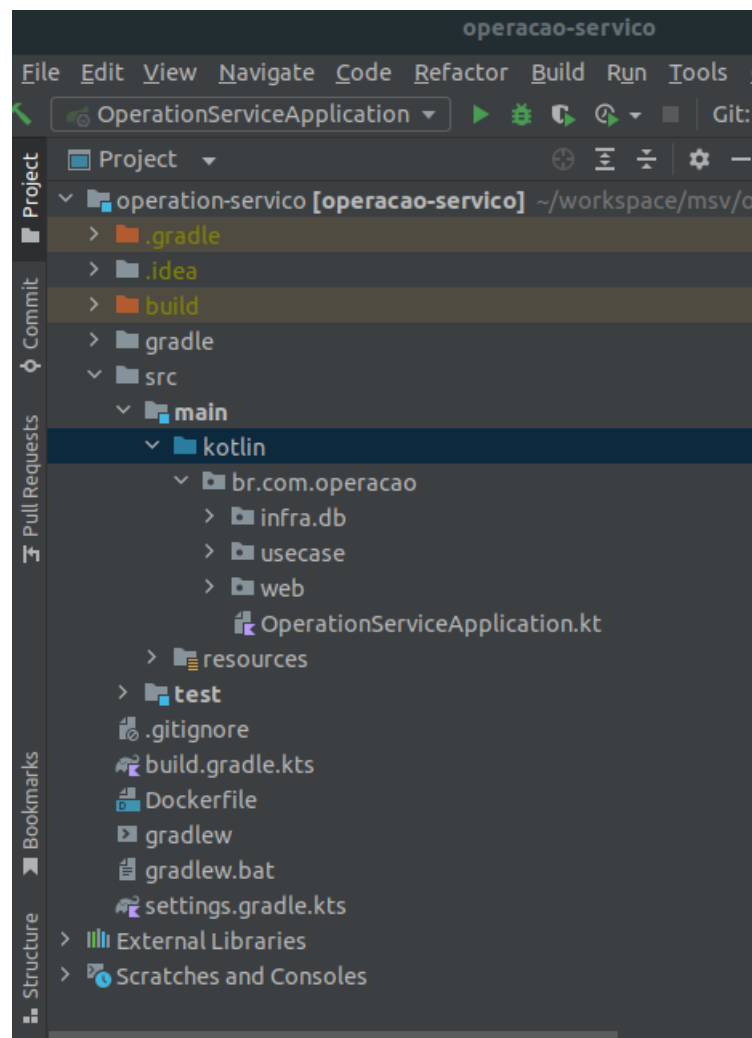


Figura 17 – Serviço de Operações

O armazenamento dos registros no banco de dados é feito pela classe de repositório que fica dentro do pacote *infra.db*, seguindo a classe de entidade, localizada dentro do pacote *entity*. Os casos de uso e as classes de modelo da aplicação estão dentro do pacote

usecase. E representando os recursos e classes que representam a requisição e a resposta do serviço estão no pacote *web*.

4.4 Interface do Sistema

A interface do sistema foi desenvolvida utilizando o *Angular CLI*. Ao gerar um projeto novo com o *framework*, os arquivos são divididos em pacotes conforme o seu padrão. Neste projeto não foi alterada sua estrutura, apenas foram criados novos componentes para implementar a interface. A figura abaixo mostra como o *Angular* divide a estrutura de um novo projeto gerado por ele.

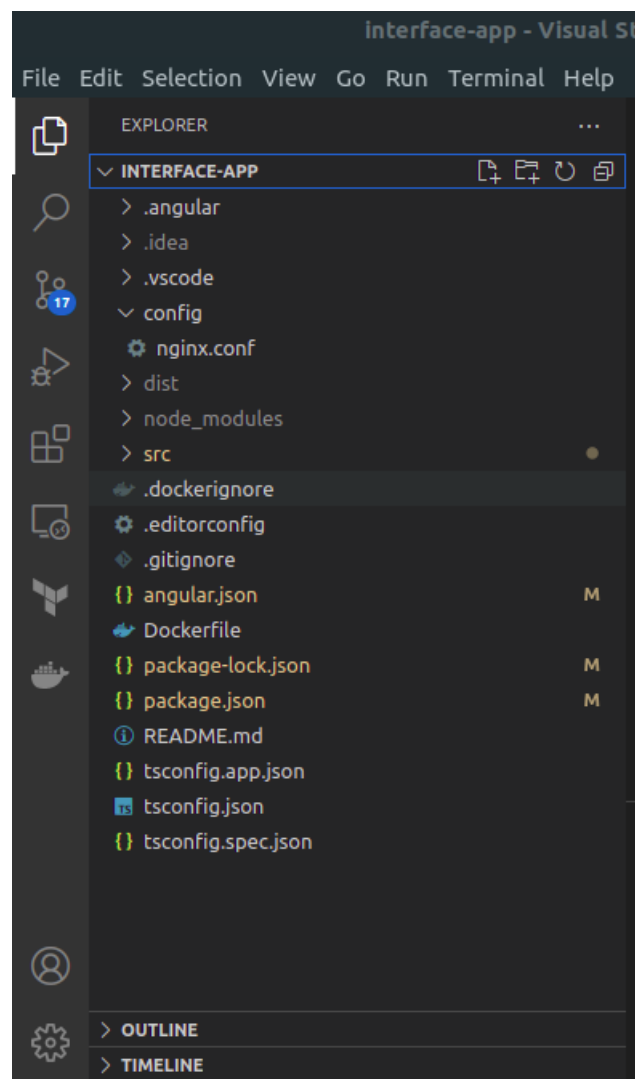


Figura 18 – Interface da Aplicação.

Nesse sistema irá ser mapeado o formulário para inserir os dados, botões para buscar e alterar possibilitando a visualização das funcionalidades do *software*.

4.5 Disponibilizando os Recursos na Nuvem

Após ter modelado e implementado os componentes arquiteturais, é preciso criar e disponibilizar os recursos na nuvem *AWS*. Para facilitar a implantação, foi provisionado os componentes através de infraestrutura como código via *Terraform*. Foi criado um projeto organizado da seguinte forma:

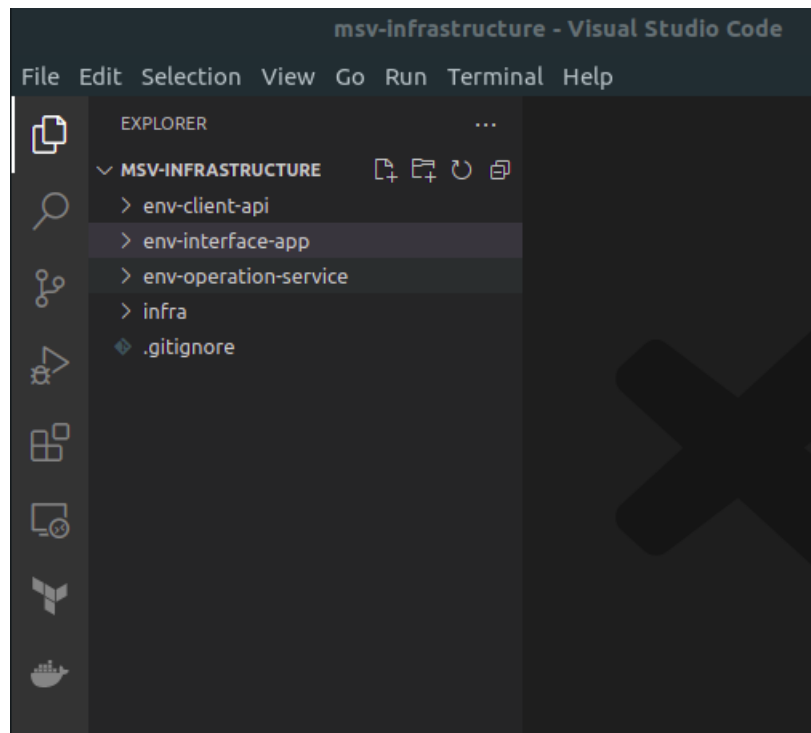


Figura 19 – Infraestrutura como Código

Na pasta *env-client-api* tem todo o código em arquivos do *terraform* para disponibilizar o que é específico da infraestrutura da aplicação de cliente. Consequentemente, na pasta *env-operation-service* e *env-interface-app* contêm o que é particular de ambos. Por fim na pasta *infra* contêm os arquivos comuns a todos os recursos provisionados.

Então para criar os recursos através desse projeto, executamos o comando do *terraform apply* que aplica todo o provisionamento escrito como código, criando os componentes na nuvem *AWS*.

De todos os componentes disponibilizados pela *AWS*, foram utilizados os: *s3*, *DynamoDB*, *RDS*, *Elastic Beanstalk*, *ECR*. Para fazer com o que os componentes se integram, foram criadas *roles*, que são regras implementadas dentro da nuvem.

Na criação do *Elastic Beanstalk*, foi utilizado a solução: *64bit Amazon Linux 2 v3.5.3 running Docker*, para permitir o *deploy* das aplicações via *Docker*. Com isso é gerada imagens *Docker*, depois submetidas ao *Amazon Elastic Container Registry* para serem referenciadas no ambiente do *Elastic beanstalk*.

Após a criação da infraestrutura via código, pode-se visualizar através do console da AWS. Para isso, deve-se fazer o *login* na plataforma e buscar os componentes que deseja visualizar. No caso das aplicações contruídas, quando é acessado o componente *Elastic Beanstalk* deverá aparecer as aplicações conforme a imagem abaixo:

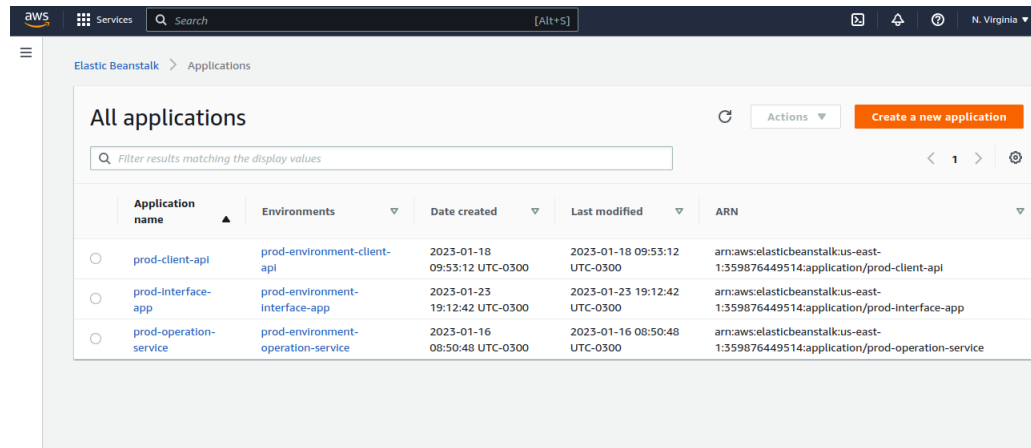


Figura 20 – Aplicações no Elastic Beanstalk

Para visualizar a tabela do *DynamoDB*, é acessado o o componente do mesmo, também via console e selecionar a tabela:

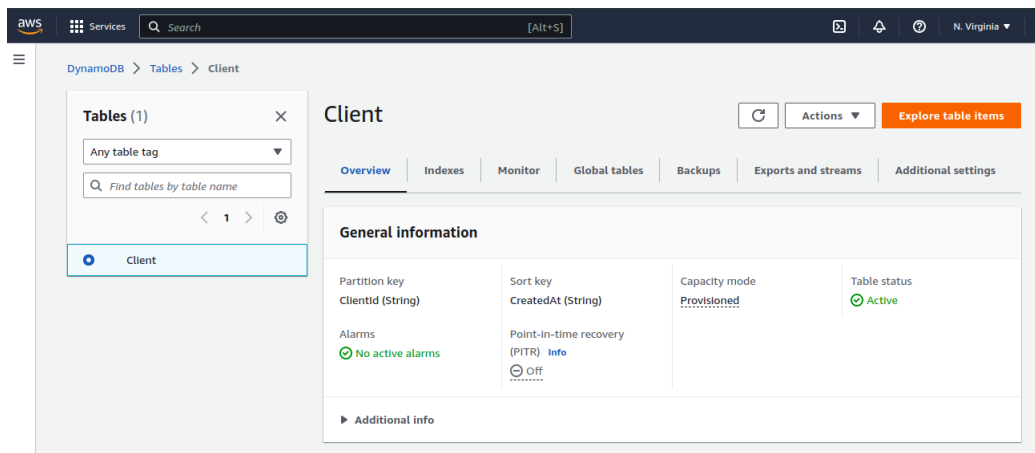


Figura 21 – Tabela do DynamoDB

Quando acessamos o *s3* deverá aparecer os seguintes:

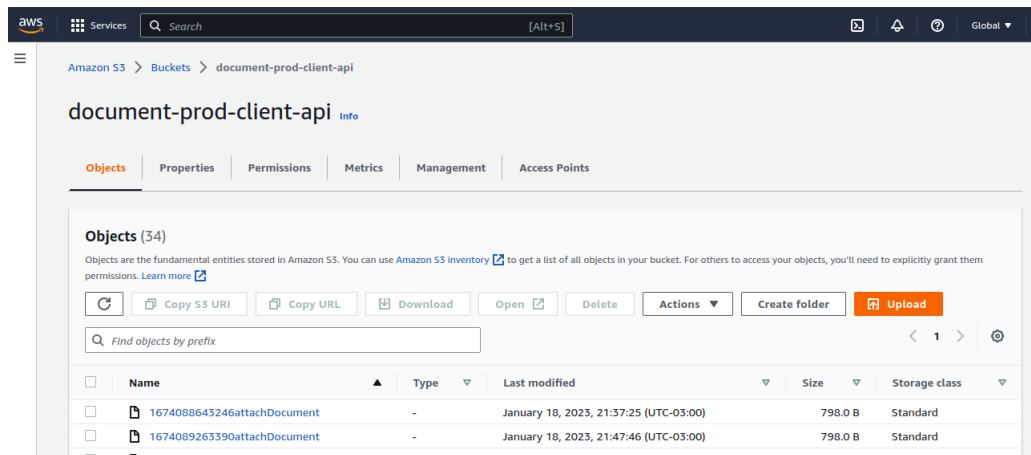


Figura 22 – Documentos do Cliente

Já o banco de dados *Postgres* é acessado através do *RDS*:

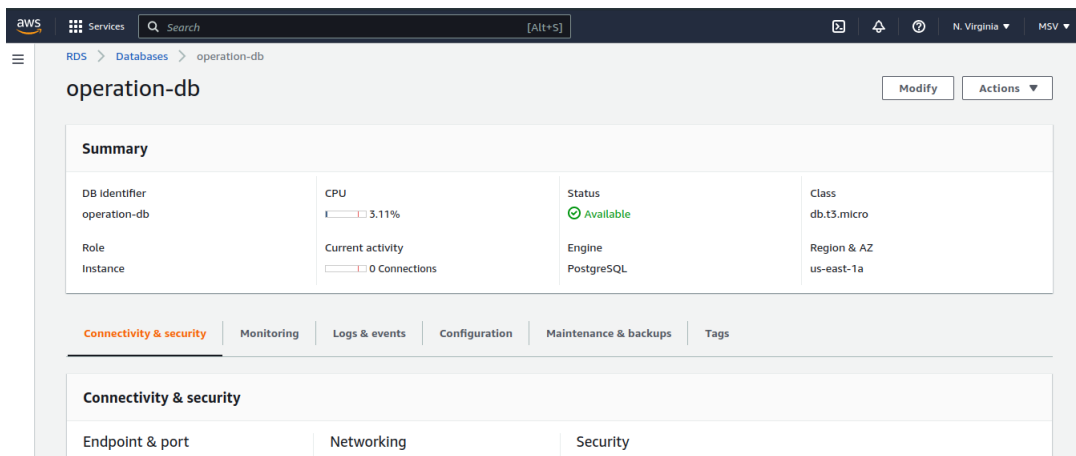


Figura 23 – Banco de Dados Operação.

E então as instancias das aplicações estão sendo executadas no *EC2*

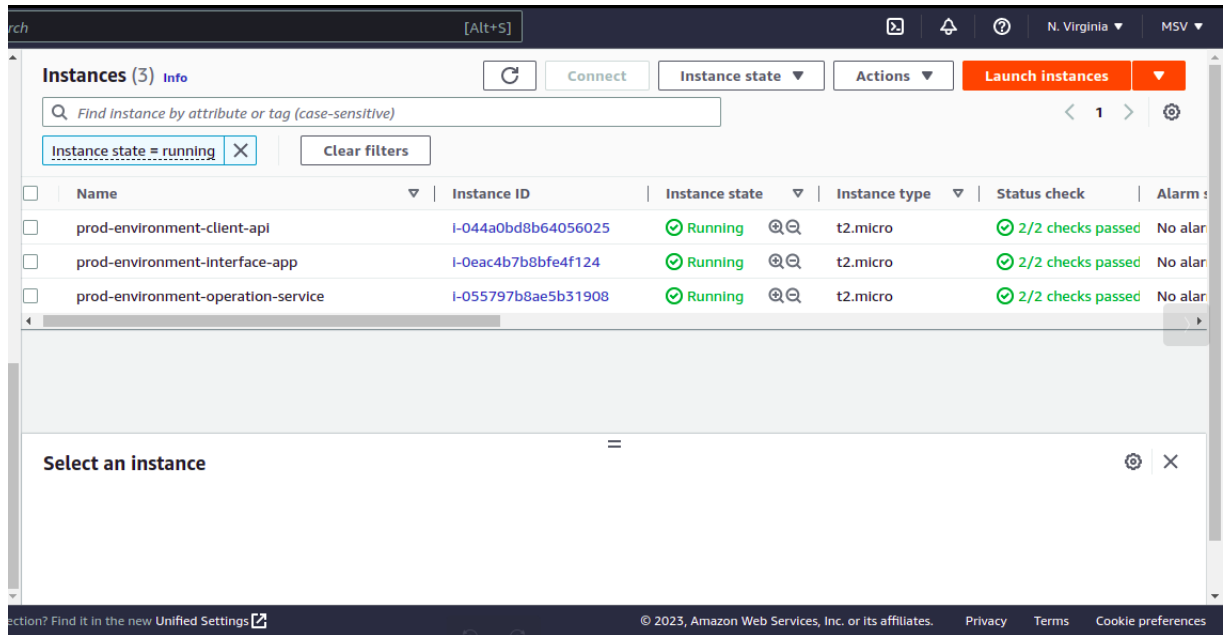


Figura 24 – Instâncias em execução.

4.6 Executando aplicação com a Interface

Com as aplicações e os recursos já sendo executados na nuvem, pode-se executar os casos de uso propostos.

- Um dos casos de uso propostos, foi possibilitar a criação de um cadastro de cliente para que futuramente possa atualizar e buscar os dados do mesmo. Os passos para realizar o cadastro são, abrir a página inicial da interface através do endereço gerado pelo provedor de nuvem AWS, conforme a imagem abaixo e clicar no botão **Cadastrar o Cliente**:

The screenshot shows a web interface for client registration. At the top, there is a blue header bar. Below it, four navigation buttons are displayed: 'Cadastrar Cliente >', 'Buscar Cliente >', 'Consultar Situação >', and 'Consultar Operações >'. The main form area contains a grid of input fields for client information:

Nome João	Número do Documento 12345	
Endereço Avenida Principal	Número 100	Bairro Centro
Cidade Uberlândia	Estado Minas Gerais	CEP 3840000

Below the grid is a text area for 'Observação do Cliente' containing the text 'Cliente com grande potencial.' and a small edit icon. Underneath is a file upload section with a 'Choose Files' button, the filename 'Documento.png', and a text input field containing 'Documento.png'. At the bottom of the form is a blue button labeled 'Criar Cliente'.

Figura 25 – Página de Cadastro.

- Para disponibilizar os dados do cadastro de um cliente, é só acessar a página inicial da interface através do endereço gerado pelo provedor de nuvem *AWS* e clicar em: **Buscar Cliente** conforme a imagem abaixo:

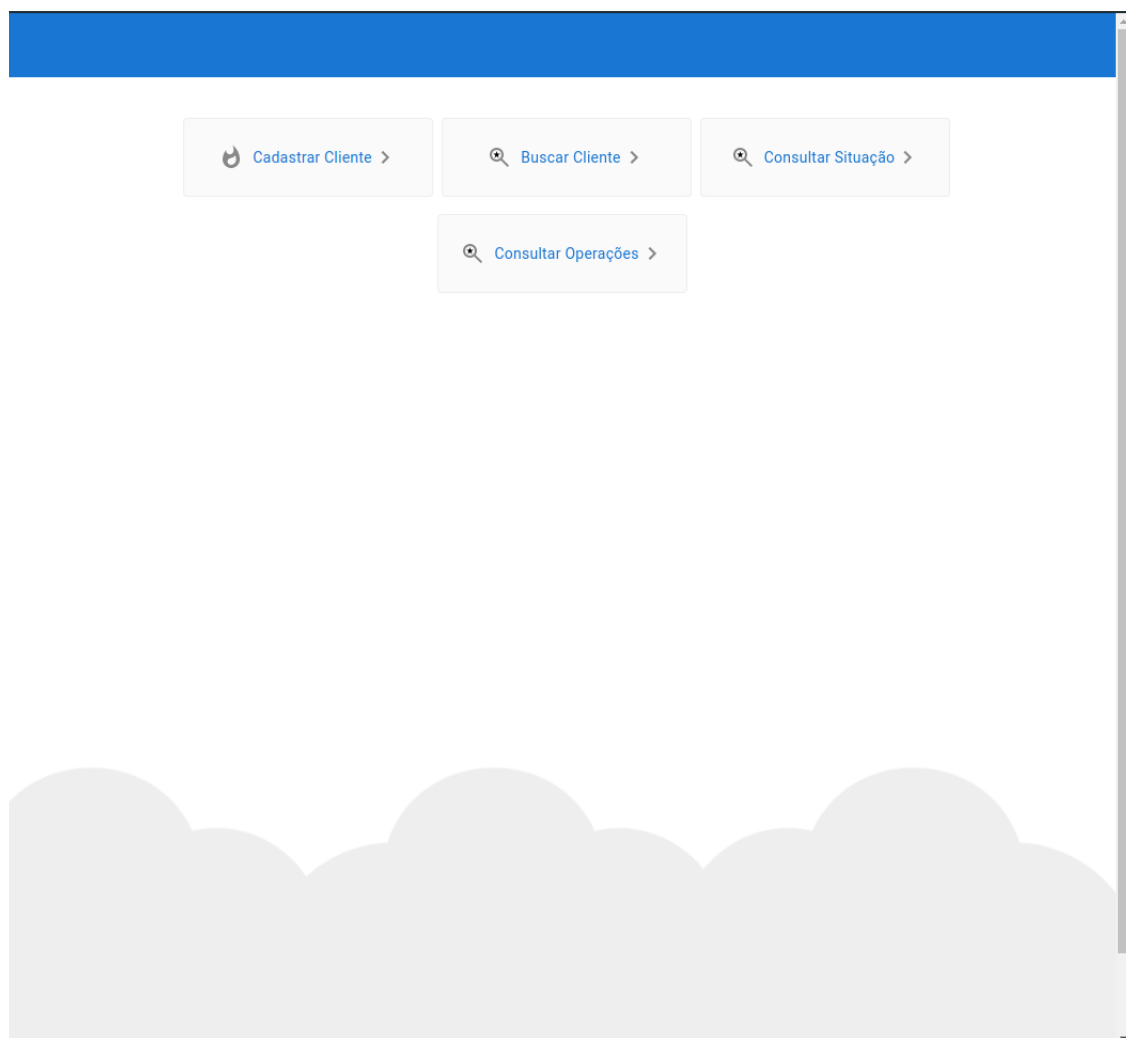


Figura 26 – Página Inicial da Interface.

Após isso irá aparecer um campo para a busca através do número do documento do cliente cadastrado. Assim pode-se digitar o número e realizar a busca. Veja na imagem a seguir:

Cadastrar Cliente >

Buscar Cliente >

Consultar Situação >

Consultar Operações >

Número do Documento
12345

Buscar

Nome
João

Número do Documento
12345

Endereço
Avenida Principal

Número
100

Bairro
Centro

Cidade
Uberlândia

Estado
Minas Gerais

CEP
3840000

Observação do Cliente
Cliente com grande potencial.

Endereço Do Documento
document-prod-client-api.s3.amazonaws.com/1675114932821attachDocument

Atualizar Cliente

Figura 27 – Página de Busca.

- Em seguida, ao buscar o cliente, é permitido alterar os dados do mesmo com a exceção do documento anexado. No exemplo é alterado o nome de João para Maria do cliente. Com isso pode-se alterar os campos mostrados na tela e clicar no botão **Atualizar Cliente**.

Cadastrar Cliente > Buscar Cliente > Consultar Situação >

Consultar Operações >

Número do Documento
12345 **Buscar**

Nome Maria	Número do Documento 12345	
Endereço Avenida Principal	Número 100	Bairro Centro
Cidade Uberlândia	Estado Minas Gerais	CEP 3840000
Observação do Cliente Cliente com grande potencial.		
Endereço Do Documento document-prod-client-api.s3.amazonaws.com/1675114932821attachDocument		

Atualizar Cliente

Figura 28 – Página de Atualizar Cliente.

- Uma funcionalidade no domínio é permitir busca a situação do cliente, no caso de uso é feita a busca e retornando ATIVO ou INATIVO. Com a página da interface aberta, basta clicar em **Consultar Situação** e preencher com o número de documento do cliente. Feito isso, será retornado conforme a imagem:

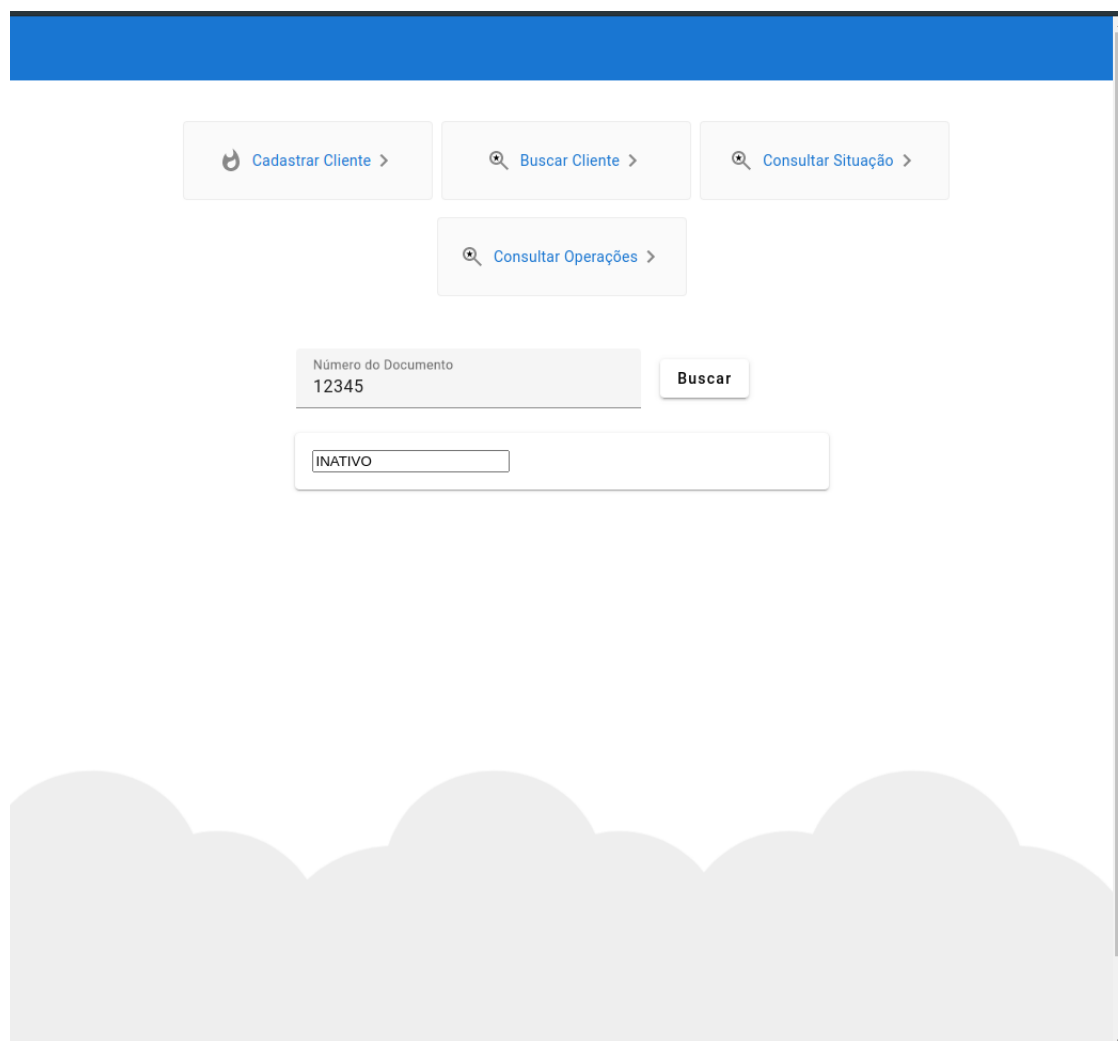


Figura 29 – Página de Situação do Cliente.

- E por fim, para consultar as operações realizadas pelo administrador sobre o cadastro, basta acessar a página inicial e clicar no botão **Consultar Operações**, fazendo a busca também por número de documento do cliente, Segue imagem com as operações:

Cadastrar Cliente >

Buscar Cliente >

Consultar Situação >

Consultar Operações >

Número do Documento
12345

Buscar

Ação	Número de Documento	Data da Operação
CREATION	12345	2023-01-30T18:42:15.023000227
UPDATE	12345	2023-01-30T18:53:57.291065330

Figura 30 – Página de Busca de Operações do Cliente.

5 Conclusão

Este trabalho de conclusão de curso teve como objetivo demonstrar e aplicar os conceitos do *DDD* em *Cloud Computing*. Desenvolver o mesmo foi muito importante para o autor sendo mostrado os conceitos tão relevantes para o desenvolvimento de software na área de Tecnologia.

Visando essa área de atuação, a importância de se conseguir colocar o sistema na nuvem de uma maneira rápida e escalável, vem sendo desafiador a cada evolução de ferramentas e tecnologias novas. Boas práticas de desenvolvimento de *software*, como por exemplo os padrões arquiteturais e de projetos, vistos também dentro do *Domain-Driven Design* auxilia grandemente à cumprir essa demanda. Focando em deixar um domínio bem estruturado, de fácil entendimento e evolução.

A busca e acompanhamento de novas tecnologias no mercado, segue um caminho para construir componentes reutilizáveis e com baixo acoplamento, com o fim de deixar mais fácil a troca de bibliotecas, ferramentas e até mesmo os provedores de nuvem. Pois o avanço das ferramentas, resulta em concorrência entre empresas que fornecem solução robustas para o mercado. Caso o foco seja até mesmo no custo, que é comum de acontecer principalmente em provedores de plataformas em nuvem, dependendo do caso de uso, opta-se por dividir fluxos e componentes entre um ou mais provedores, conseguindo diminuir o custo e até mesmo escolher melhores recursos dentre eles.

A construção deste trabalho foi com base na elaboração e aplicação de conceitos importantes do *Domain-Driven Design*, criação e disponibilização dos componentes propostos pela arquitetura do *software* em um ambiente de nuvem, visando explorar recursos que ajudam a entregar sistemas mais rápidos em ambientes produtivos e de uma maneira que seja reaproveitados por outros domínios. O resultado dessa construção foi descrever casos de uso para que sejam possíveis de se trabalhar com os conceitos do *DDD*, construindo uma arquitetura contextual que utiliza vários recursos de provedores de nuvem, para demonstrar uma forma de entender a necessidade de negócio, modelar e implantar em um ambiente produtivo possibilitando ser escalado horizontalmente, ou seja, adicionando mais nós ao sistema.

Portanto é possível analisar a partir de todo o conteúdo deste trabalho, que o desenvolvimento aplicando conceitos arquiteturais, padrões de projetos, modelagem de *software* e boas práticas permite criar sistemas com domínios bem definidos de uma maneira bem entendível possibilitando a sua evolução. Junto a arquitetura construída, foi demonstrado vantagens de se usar computação em nuvem e uma maneira de colocar esses sistemas e recursos utilizados por eles em um ambiente produtivo, sendo disponibilizado

para acessos externos e internos entre os componentes de infraestrutura, com o custo sobre demanda, ou seja, é cobrado de acordo com o uso e também sendo gerenciado grande parte do ciclo de execução e implantação dos recursos pelo provedor de nuvem.

Para próximos trabalhos, pode-se explorar o como refatorar uma arquitetura simples ou complexa, já existente, aplicando os conceitos do *Domain-Driven Design*. Visto que este trabalho, propõe uma solução nova a ser construída. E abordar o resultado dessa refatoração em uma infraestrutura na nuvem.

Referências

- ANGULAR. **Angular**. [S.l.], 2023. Disponível em: <<https://angular.io/>>. Acesso em: 28 jan. 2023. Citado na página 29.
- ANICHE, M.; BAVOTA, G.; TREUDE, C.; GEROSA, M. A.; DEURSEN, A. van. Code smells for Model-View-Controller architectures. **Empirical Software Engineering**, Springer New York LLC, v. 23, n. 4, p. 2121–2157, 8 2018. ISSN 15737616. Disponível em: <<https://link-springer-com.ez34.periodicos.capes.gov.br/article/10.1007/s10664-017-9540-2>>. Citado na página 9.
- AWS. **What is Cloud Computing**. [S.l.], 2022. Disponível em: <https://aws.amazon.com/what-is-cloud-computing/?nc1=h_ls>. Acesso em: 19 may. 2022. Citado 2 vezes nas páginas 10 e 18.
- AWS, A. **What is AWS**. [S.l.], 2022. Disponível em: <https://aws.amazon.com/what-is-aws/?nc1=h_ls>. Acesso em: 11 oct. 2022. Citado 2 vezes nas páginas 19 e 29.
- AWS-BENSTALK. **What is AWS Elastic Beanstalk?** [S.l.], 2022. Disponível em: <<https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/Welcome.html>>. Acesso em: 08 nov. 2022. Citado 2 vezes nas páginas 6 e 20.
- AWS-DYNAMODB. **Amazon DynamoDB**. [S.l.], 2022. Disponível em: <<https://aws.amazon.com/pt/dynamodb>>. Acesso em: 05 nov. 2022. Citado 2 vezes nas páginas 6 e 21.
- AWS-EC2. **Amazon EC2**. [S.l.], 2022. Disponível em: <<https://aws.amazon.com/pt/ec2>>. Acesso em: 05 nov. 2022. Citado na página 21.
- AWS-ELB. **Elastic Load Balancing**. [S.l.], 2022. Disponível em: <<https://aws.amazon.com/pt/elasticloadbalancing/>>. Acesso em: 05 nov. 2022. Citado na página 22.
- AWS-RDS. **Amazon Relational Database Service**. [S.l.], 2022. Disponível em: <<https://aws.amazon.com/pt/rds>>. Acesso em: 05 nov. 2022. Citado 3 vezes nas páginas 6, 25 e 26.
- AWS-S3. **Amazon S3**. [S.l.], 2022. Disponível em: <<https://aws.amazon.com/pt/s3>>. Acesso em: 05 nov. 2022. Citado 2 vezes nas páginas 6 e 23.
- AWS-SNS. **Amazon Simple Notification Service**. [S.l.], 2022. Disponível em: <<https://aws.amazon.com/pt/sns/?whats-new-cards.sort-by=item.additionalFields.postDateTime&whats-new-cards.sort-order=desc>>. Acesso em: 05 nov. 2022. Citado 3 vezes nas páginas 6, 23 e 24.
- AWS-SQS. **Amazon Simple Queue Service**. [S.l.], 2022. Disponível em: <<https://aws.amazon.com/pt/sqs>>. Acesso em: 05 nov. 2022. Citado 3 vezes nas páginas 6, 24 e 25.
- AZURE. **Azure**. [S.l.], 2023. Disponível em: <<https://azure.microsoft.com/pt-br/>>. Acesso em: 05 fev. 2023. Citado na página 19.

- BENSTALK, A. **AWS Elastic Beanstalk Documentation**. [S.l.], 2022. Disponível em: <https://docs.aws.amazon.com/pt_br/elastic-beanstalk/index.html>. Acesso em: 11 oct. 2022. Citado na página 20.
- BRIAN W. KERNIGHAN; ROB PIKE. **The UNIX Programming Environment**. 1984. Disponível em: <<http://dcis.uohyd.ac.in/~apcs/itw/UNIXProgrammingEnvironment.pdf>>. Acesso em: 27 may. 2022. Citado na página 9.
- COTTON IW; GREATORIX FS; JR. DATA STRUCTURES AND TECHNIQUES FOR REMOTE COMPUTER GRAPHICS. v. 33, n. pt 1, p. 533–544, 1968. Acesso em: 27 may. 2022. Citado na página 9.
- DIKAIKAKOS, M. D.; KATSAROS, D.; MEHRA, P.; PALLIS, G.; VAKALI, A. Cloud computing: Distributed internet computing for IT and scientific research. **IEEE Internet Computing**, v. 13, n. 5, p. 10–11, 2009. ISSN 10897801. Acesso em: 20 jun. 2022. Citado na página 19.
- ERIC EVANS. **Domain-Driven Design Tackling Complexity in the Heart of Software**. 3. ed. [S.l.: s.n.], 2016. Acesso em: 27 may. 2022. Citado 6 vezes nas páginas 6, 9, 13, 14, 15 e 17.
- EVANS, E. Domain--Driven Design Reference Definitions and Pattern Summaries. 2015. Disponível em: <<http://creativecommons.org/licenses/by/4.0/.ii>>. Acesso em: 27 may. 2022. Citado 2 vezes nas páginas 9 e 15.
- FOWLER, M. **DDD__Aggregate**. [S.l.], 2013. Disponível em: <https://martinfowler.com/bliki/DDD__Aggregate.html>. Acesso em: 27 may. 2022. Citado 2 vezes nas páginas 16 e 17.
- GITHUB. **GitHub**. [S.l.], 2023. Disponível em: <<https://github.com/>>. Acesso em: 28 jan. 2023. Citado na página 29.
- GOOGLE. **Google Cloud**. [S.l.], 2023. Disponível em: <<https://cloud.google.com/?hl=pt-br>>. Acesso em: 05 fev. 2023. Citado na página 19.
- HASHICORP. **Terraform**. [S.l.], 2023. Disponível em: <<https://www.terraform.io/>>. Acesso em: 28 jan. 2023. Citado na página 30.
- IBM. **IBM Cloud**. [S.l.], 2023. Disponível em: <<https://cloud.ibm.com/>>. Acesso em: 05 fev. 2023. Citado na página 19.
- LANG, K. **Kotlin Overview**. [S.l.], 2023. Disponível em: <<https://kotlinlang.org/docs/getting-started.html>>. Acesso em: 28 jan. 2023. Citado na página 29.
- LE, D. M.; DANG, D.-H.; NGUYEN, V.-H. Generative software module development for domain-driven design with annotation-based domain specific language. **Information and Software Technology**, v. 120, p. 106239, 2020. Disponível em: <<https://doi.org/10.1016/j.infsof.2019.106239>>. Acesso em: 31 jan. 2023. Citado na página 11.
- MARSTON, S.; LI, Z.; BANDYOPADHYAY, S.; ZHANG, J.; GHALSASI, A. Cloud computing — The business perspective. **Decision Support Systems**, North-Holland, v. 51, n. 1, p. 176–189, 4 2011. ISSN 0167-9236. Acesso em: 31 jan. 2023. Citado na página 11.

- _____. Cloud computing — The business perspective. **Decision Support Systems**, North-Holland, v. 51, n. 1, p. 176–189, 4 2011. ISSN 0167-9236. Acesso em: 20 jun. 2022. Citado 2 vezes nas páginas 18 e 19.
- OCEAN, D. **Digital Ocean**. [S.l.], 2023. Disponível em: <<https://www.digitalocean.com/>>. Acesso em: 05 fev. 2023. Citado na página 19.
- ORACLE. **Oracle Cloud**. [S.l.], 2023. Disponível em: <<https://www.oracle.com/br/cloud>>. Acesso em: 05 fev. 2023. Citado na página 19.
- PENCHIKALA, S. **Domain Driven Design and Development In Practice**. [S.l.], 2022. Disponível em: <<https://www.infoq.com/articles/ddd-in-practice/>>. Acesso em: 19 may. 2022. Citado 2 vezes nas páginas 13 e 16.
- RADEMACHER, F.; SORGALLA, J.; SACHWEH, S. Challenges of domain-driven microservice design: A model-driven perspective. **IEEE software**, IEEE, Los Alamitos, v. 35, n. 3, p. 36–43, 2018. ISSN 0740-7459. Acesso em: 20 jun. 2022. Citado 4 vezes nas páginas 9, 13, 16 e 17.
- _____. Challenges of domain-driven microservice design: A model-driven perspective. **IEEE Software**, IEEE Computer Society, v. 35, n. 3, p. 36–43, 5 2018. ISSN 07407459. Acesso em: 31 jan. 2023. Citado na página 10.
- SPRING. **Spring Framework**. [S.l.], 2023. Disponível em: <<https://spring.io/projects/spring-framework>>. Acesso em: 28 jan. 2023. Citado na página 29.
- STÉPHANE NICOLL; DAVE SYER; MADHURA BHAVE. **Spring Guide**. [S.l.], 2022. Disponível em: <<https://docs.spring.io/initializr/docs/current/reference/html/>>. Acesso em: 19 may. 2022. Citado na página 9.
- VOUK, M. A. **Cloud computing—issues, research and implementations - Gale Academic OneFile**. [S.l.], 2008. Disponível em: <<https://go-gale.ez34.periodicos.capes.gov.br/ps/i.do?p=AONE&u=capes&id=GALE|A193140927&v=2.1&it=r>>. Acesso em: 20 jun. 2022. Citado na página 18.
- XU, X. From cloud computing to cloud manufacturing. **Robotics and Computer-Integrated Manufacturing**, Pergamon, v. 28, n. 1, p. 75–86, 2 2012. ISSN 0736-5845. Acesso em: 20 jun. 2022. Citado 2 vezes nas páginas 11 e 18.

ANEXO A – Projeto de Infraestrutura como Código

Para disponibilizar a infraestrutura na *AWS* como código, foi desenvolvido o projeto abaixo na linguagem *Terraform* para provisionar os recursos utilizados. A arquitetura de pastas para reutilizar os módulos codificados, foi dividida da seguinte maneira:

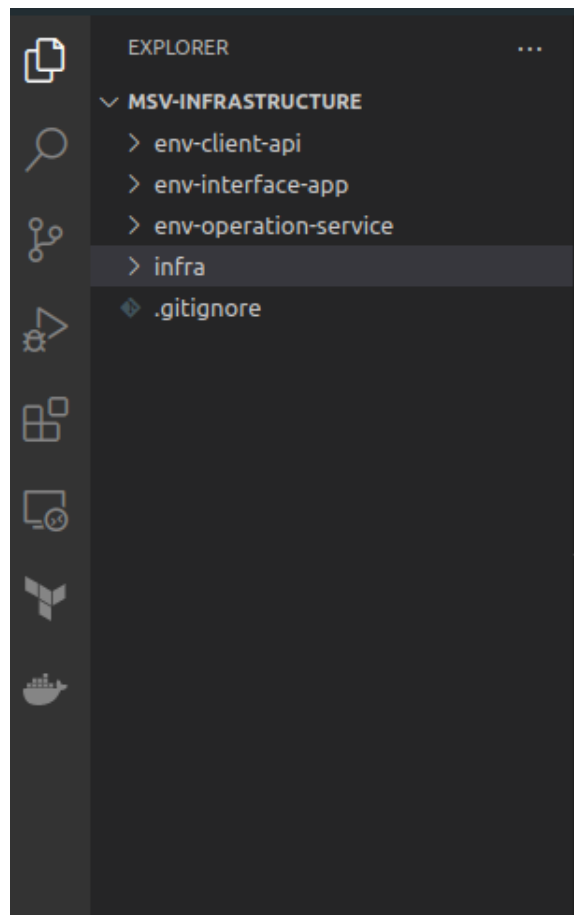


Figura 31 – Arquitetura Projeto de Infraestrutura.

No pacote *env-client-api* estão todos os arquivos de recursos que são utilizados exclusivamente para a api de Cliente, como o *DynamoDB* utilizado para armazenar os dados de Cliente e o *s3* para armazenar o documento anexado no cadastro de Cliente. Já no pacote *env-interface-app* estão os recursos necessários para disponibilizar a interface da solução na nuvem. E no pacote *env-operation-service* estão os arquivos que são de uso específico da aplicação de Operações, no caso o *RDS* com provisionado com o banco de dados *PostGrees*. Conforme imagem abaixo:

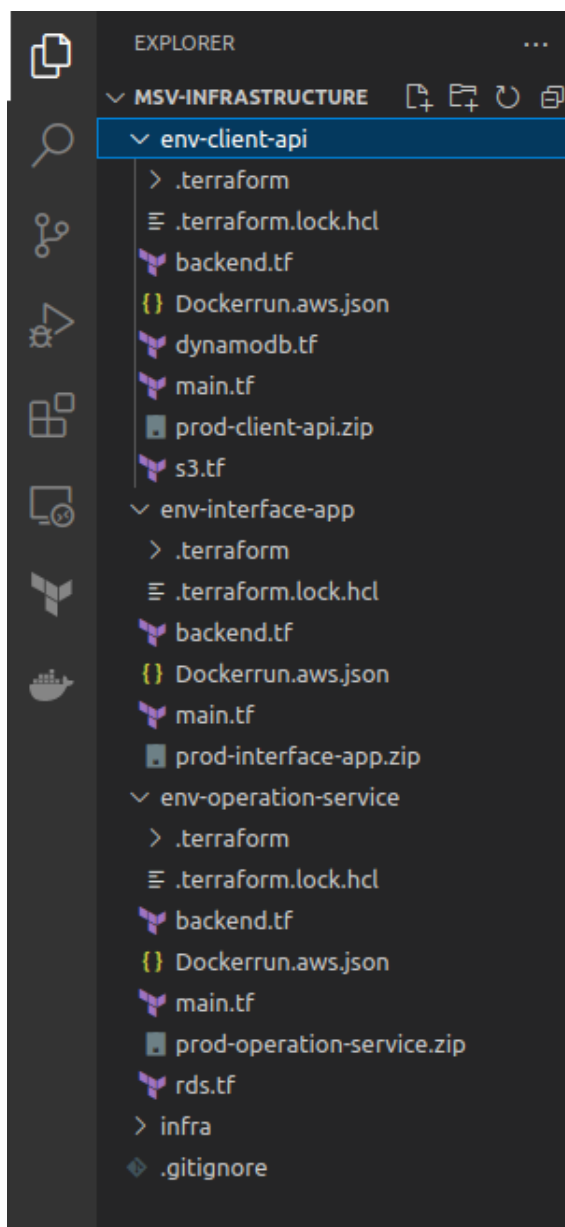


Figura 32 – Arquitetura Da Aplicações.

Por fim no pacote de infra, estão os arquivos necessários para hospedar os arquivos de estado do *Terraform* e os módulos com os recursos necessários para provisionar o *Elastic Beanstalk* que é o ambiente onde as aplicações estão sendo executadas.

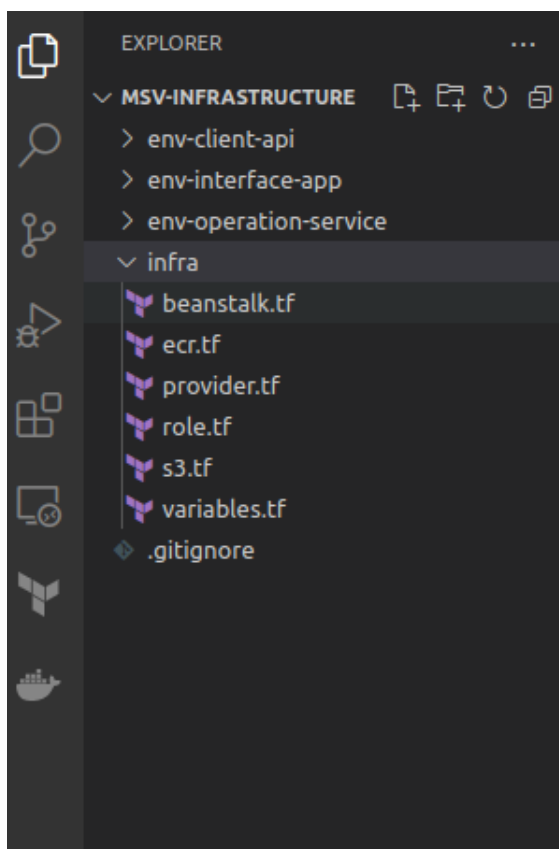


Figura 33 – Arquitetura Pacote Infra.

ANEXO B – Repositórios para Colaboração

Os repositórios das aplicações de Cliente, Operação, Interface e até mesmo o de Infraestrutura estão disponibilizados no *GitHub* como públicos, para futuras evoluções e possibilitando de colaborações para outros usuários da ferramenta. Segue link de acesso abaixo:

- [API de Cliente](#)
- [API de Operações](#)
- [Interface](#)
- [Infraestrutura](#)