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

Augusto Hideki Uda

Desenvolvimento de um sistema automatizado de aquisição de dados da plataforma Booking

Florianópolis
2024

Augusto Hideki Uda

Desenvolvimento de um sistema automatizado de aquisição de dados da plataforma Booking

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. Carlos Barros Montez, Dr.
Supervisor: Bruno Benetti, Eng.

Florianópolis
2024

Ficha de identificação da obra

A ficha de identificação é elaborada pelo próprio autor.
Orientações em:
http://portalbu.ufsc.br/ficha

Augusto Hideki Uda

Desenvolvimento de um sistema automatizado de aquisição de dados da plataforma Booking

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, 10 de julho de 2024.

Prof. Marcelo de Lellis Costa de Oliveira, Dr.
Coordenador do Curso

Banca Examinadora:

Documento assinado digitalmente
Carlos Barros Montez
Data: 28/07/2024 19:12:44-0300
CPF: ***.035.027-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Carlos Barros Montez, Dr.
Orientador
UFSC/CTC/DAS

Documento assinado digitalmente
BRUNO EDUARDO BENETTI
Data: 29/07/2024 14:16:41-0300
CPF: ***.335.709-**
Verifique as assinaturas em https://v.ufsc.br

Bruno Eduardo Benetti, Me.
Supervisor
Instituição Seazone

Prof. Thiago Raulino Dal Pont, Me.
Avaliador
UFSC/CTC/DAS

Prof. Hector Bessa Silveira, Dr.
Presidente da Banca
UFSC/CTC/DAS

Este trabalho é dedicado aos meus amigos e família,
pelo suporte e carinho durante toda essa jornada.

AGRADECIMENTOS

A conclusão deste projeto representa o termino de um capítulo importante da minha vida, em que muitas pessoas fizeram parte fundamental de seu desdobramento, oferecendo apoio, incentivo e orientação ao longo dessa jornada.

Primeiramente, sou extremamente grato à minha família, cujo apoio incondicional serviu de alicerce sólido, assegurando de que eu me mantivesse firme nos momentos mais desafiadores. Em especial, gostaria de registrar minha profunda gratidão à minha querida mãe Helena, que sempre colocou meu bem-estar e desenvolvimento pessoal como prioridades, nunca me faltando com amor em todos esses anos de vida.

Também sou grato aos meus colegas de trabalho, principalmente ao time de dados da Seazone. André Crescenzo, Artur Brito, Francisco Burigo, Lucas Abel, Márcio Fazolin, Patrick de Souza e a todos que já passaram pela equipe durante minha estadia, sou grato pela parceria e troca de experiência que foram fundamentais para realização deste trabalho.

Ao Departamento de Automação e Sistemas (DAS) da Universidade Federal de Santa Catarina (UFSC), agradeço por fornecer esse ambiente acadêmico estimulante, desafiador e acolhedor, que é o curso de Engenharia de Controle e Automação. Em particular, agradeço ao professor Carlos Montez por sua orientação e disponibilidade durante o período de confecção desta monografia.

Por fim, expresso minha sincera gratidão aos meu amigos mais próximos, com os quais tenho a felicidade de ter compartilhado momentos de alegria, desafio e crescimento, que sempre estiveram ao meu lado oferecendo apoio e companheirismo constantes.

A todos que fizeram parte direta ou indireta deste percurso ao longo dos anos de graduação, muito obrigado.

Em pleno inverno, aprendia por fim
que existia em meu ser um verão invencível
(Camus, 1954)

DECLARAÇÃO DE PUBLICIDADE

Florianópolis, 25 de junho de 2024.

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: 29/07/2024 14:17:28-0300
CPF: ***.335.709-**
Verifique as assinaturas em https://v.ufsc.br

Bruno Benetti
Seazone Serviços LTDA

RESUMO

Esta monografia trata do projeto e implementação de um sistema de aquisição automatizada de dados para a Seazone, uma empresa brasileira especializada na administração de imóveis para aluguel por temporada. O projeto visa solucionar a falta de dados relevantes para análises de mercado provenientes do Booking, uma grande referência na indústria das Online Travel Agencies (OTAs). A coleta das informações é realizada com o uso de web Scrapers, programas especializados na coleta de informações públicas espalhadas pela internet. O sistema de aquisição é inserido na plataforma de serviços de nuvem AWS, onde é feito amplo proveito de recursos de computação distribuída, processamento e armazenamento de dados. A implementação do projeto é realizada através de uma metodologia arquitetada para o desenvolvimento sistemático de web scrapers, incluindo etapas de exploração dos sites, testes remotos e integração com a nuvem. Os dados coletados pelo sistema são inseridos no Data Lake da Seazone, local em que são arquivados para o uso posterior em tomadas de decisão e processos analíticos da empresa. O projeto alcançou grandes avanços no desenvolvimento da infraestrutura de coleta de dados da empresa, sendo capaz de extrair semanalmente informações relevantes acerca de um grande número de listagens do Booking.

Palavras-chave: Aquisição de dados. Serviços de nuvem. Web scraping. Booking.

ABSTRACT

This monograph deals with the design and implementation of an automated data acquisition system for Seazone, a Brazilian company specializing in the management of vacation rental properties. The project aims to solve the lack of relevant data for market analysis from Booking, a major player in the Online Travel Agency (OTA) industry. The information is collected using Web Scrapers, which are programs specialized in collecting public information spread across the Internet. The acquisition system is deployed on the AWS cloud services platform, where extensive use is made of distributed computing, processing and data storage resources. The project is implemented using a methodology designed for the systematic development of Web Scrapers, including stages of site exploration, remote testing and integration with the cloud. The data collected by the system is dumped into Seazone's Data Lake, where it is archived for later use in the company's decision-making and analytical processes. The project has made great strides in developing the company's data collection infrastructure, being able to extract relevant information on a weekly basis from a large number of Booking listings.

Keywords: Data acquisition. Cloud services. Web scraping. Booking.

LISTA DE FIGURAS

Figura 1 – Captura de tela da página de um hotel no Booking.com. 20
Figura 2 – Estrutura de arquivo Parquet. 23
Figura 3 – Fluxograma do processo generalizado de scrapagem. 24
Figura 4 – Gráfico da proporção de trafego na internet entre humanos, robôs bons e robôs maus de 2013 a 2023. 26
Figura 5 – Representação de várias aplicações executadas simultaneamente através de múltiplos processos, núcleos e computadores. 33
Figura 6 – Fluxograma da arquitetura simplificada do data lake da Seazone. . . 36
Figura 7 – Captura de tela da aba de avaliações do Booking. 44
Figura 8 – Captura de tela dos comentários na aba de avaliações do Booking. 45
Figura 9 – Captura de tela das regras da casa de uma listagem do Booking. . . 46
Figura 10 – Captura de tela da aba de detalhes de imóvel no Booking. 47
Figura 11 – Captura de tela do calendário de reservas do Booking. 48
Figura 12 – Captura de tela da página da Stays com conexão aos anúncios da Seazone no Booking. 49
Figura 13 – Fluxograma da arquitetura em nuvem da solução proposta. 50
Figura 14 – Fluxograma da arquitetura proposta para a extração de dados do Booking. 51
Figura 15 – Código representativo da lógica simplificada de um *scraper* do Booking. 56
Figura 16 – Exemplo de um JSON de *inputs* contendo três imóveis do Booking. 56
Figura 17 – Segmento de código ilustrando o funcionamento do BeatifulSoup na extração de dados. 57
Figura 18 – Configurações da fila SQS de *inputs* do *scraper* de detalhes. 60
Figura 19 – Fluxograma do funcionamento da classe SQSConsumer. 61
Figura 20 – Organização dos arquivos do projeto no repositório Pipe-scrapers. . 62
Figura 21 – Trecho de código do *script* booking.py. 63
Figura 22 – Captura de tela com as principais configurações dos Firehose Delivery Streams de *scrapers* do Booking. 64
Figura 23 – Fluxograma do funcionamento dos testes pontuais em EC2. 65
Figura 24 – Máquina de estados do *scraper* de comentários. 68
Figura 25 – Captura de tela com as partições dos dados de detalhes. 72
Figura 26 – Captura do envio de dados pelo Slack da empresa. 72
Figura 27 – Gráfico da porcentagem de valores nulos em colunas da tabela de comentários. 74
Figura 28 – Gráfico da porcentagem de valores nulos em colunas da tabela de regras da casa. 75

Figura 29 – Gráfico da porcentagem de valores nulos em colunas da tabela de avaliações. 76

LISTA DE TABELAS

Tabela 1 – Tabela de alternativas de combinações de métodos viáveis de busca e formato dos dados retornados. 54
Tabela 2 – Análises de integridade dos dados adquiridos. 73
Tabela 3 – Tabela de comparação de eficiência entre os *scrapers* do Booking e o de detalhes do Airbnb. 77

SUMÁRIO

1 INTRODUÇÃO 16
1.1 EMPRESA SEAZONE 16
1.2 MOTIVAÇÃO 17
1.3 ORGANIZAÇÃO DO DOCUMENTO 18
2 FUNDAMENTAÇÃO TEÓRICA 19
2.1 BOOKING.COM 19
2.2 STAYS 20
2.3 APIS 21
2.4 FORMATOS DE DADOS RELEVANTES 21
2.4.1 JSON 22
2.4.2 HTML 22
2.4.3 Parquet 22
2.5 WEB SCRAPING 23
2.5.1 Etapas de *scrapagem* 23
2.5.2 Métodos relevantes de scrapagem 24
2.5.3 Empecilhos de scrapagem 26
2.6 DATA LAKE 27
2.7 DOCKER 27
2.8 AMAZON WEB SERVICES 28
2.8.1 S3 28
2.8.2 ECS 29
2.8.3 ECR 30
2.8.4 VPC 30
2.8.5 EC2 30
2.8.6 SQS 31
2.8.7 Amazon Kinesis Data Firehose 31
2.8.8 Amazon Athena 31
2.8.9 AWS Glue 31
2.8.10 Step Functions 32
2.9 COMPUTAÇÃO CONCORRENTE 32
2.9.1 Tarefas I/O-bound e CPU-bound 33
2.9.2 Paralelismo de dados 33
2.10 PYTHON E BIBLIOTECAS RELEVANTES 34
2.11 ARQUITETURA DO DATA LAKE DA SEAZONE 35
2.11.1 Armazenamento de dados 35
2.11.2 Aquisição e ingestão de dados 37
2.11.3 Entrega de dados 37

2.11.4 Contas AWS da Seazone . . . 37
3 DESCRIÇÃO DOS PROBLEMAS E REQUISITOS TÉCNICOS . . . 39
3.1 FALTA DE DADOS EXTERNOS DO BOOKING . . . 39
3.2 FALTA DE DADOS INTERNOS DO BOOKING . . . 39
3.3 DIFICULDADE DE ACESSO AUTOMÁTICO À CONTA DO BOOKING 40
3.4 DETECÇÃO DE ROBÔS . . . 40
3.5 REQUISITOS TÉCNICOS . . . 40
4 EXPLORAÇÃO DAS PLATAFORMAS E SOLUÇÃO PROPOSTA . . 42
4.1 EXPLORAÇÃO DAS PLATAFORMAS . . . 42
4.1.1 Metodologia de exploração . . . 42
4.1.2 Avaliações do anúncio . . . 43
4.1.3 Comentários do anúncio . . . 43
4.1.4 Regras da casa do anúncio . . . 44
4.1.5 Detalhes dos imóveis . . . 44
4.1.6 Preço e disponibilidade dos imóveis . . . 45
4.1.7 URLs e IDs de anúncios da Seazone . . . 46
4.2 SOLUÇÃO PROPOSTA . . . 48
4.2.1 Arquitetura de scrapers do Booking . . . 49
5 DESENVOLVIMENTO . . . 53
5.1 METODOLOGIA DE DESENVOLVIMENTO DOS SCRAPERS DO BOOKING . . . 53
5.2 SELEÇÃO DOS MÉTODOS DE WEB *SCRAPING* . . . 54
5.3 IMPLEMENTAÇÃO LOCAL DOS PROTÓTIPOS DOS *SCRAPERS* . 55
5.3.1 *Scrapers* com extração de dados em HTML . . . 56
5.3.2 *Scraper* de detalhes . . . 57
5.3.3 Informações coletadas . . . 58
5.4 IMPLEMENTAÇÃO DO *SCRIPT* DE REFIL DE INPUTS . . . 58
5.5 INTEGRAÇÃO DOS *SCRAPERS* LOCAIS COM A NUVEM . . . 59
5.5.1 *Framework* dos *scrapers* . . . 59
5.5.2 Variáveis importantes . . . 61
5.5.3 Organização de arquivos . . . 62
5.5.4 Criação de recursos na AWS . . . 63
5.6 TESTE DO PROCESSO EM EC2 . . . 64
5.6.1 Resultado dos testes . . . 65
5.7 IMPLEMENTAÇÃO DO SISTEMA DE WEB *SCRAPING* EM ECS . . 65
5.8 AUTOMAÇÃO COMPLETA DO SISTEMA EM PRODUÇÃO . . . 67
5.8.1 Configuração das máquinas de estado . . . 67
5.8.2 Configuração e agendamento de *crawlers* . . . 69
5.9 IMPLEMENTAÇÃO DO *SCRAPER* DA STAYS . . . 69

6 ANÁLISE DOS RESULTADOS 71
6.1 REQUISITOS FUNCIONAIS 71
6.1.1 Análises de consistência dos dados 72
6.2 REQUISITOS NÃO FUNCIONAIS 75
7 CONSIDERAÇÕES FINAIS 78
REFERÊNCIAS 80
APÊNDICE A – DADOS DO *SCRAPER* DE DETALHES 83
APÊNDICE B – DADOS DO *SCRAPER* DE COMENTÁRIOS 84
APÊNDICE C – DADOS DO *SCRAPER* DE AVALIAÇÕES 85
APÊNDICE D – DADOS DO *SCRAPER* DE REGRAS DA CASA . 86

1 INTRODUÇÃO

Este Projeto de Final de Curso (PFC) aborda um desafio significativo na indústria de aluguel por temporada: a carência de dados relevantes para análises de mercado eficazes. Neste contexto, a Seazone, uma empresa que se destaca pela gestão tecnológica de imóveis, busca aprimorar suas operações através da integração de um sistema automatizado de aquisição de dados provenientes da plataforma Booking. Este projeto visa preencher essa lacuna informacional, permitindo uma melhor compreensão do mercado e otimizando as estratégias de precificação e gestão de imóveis.

1.1 EMPRESA SEAZONE

A Seazone é uma empresa brasileira especialista no setor de administração de imóveis para aluguel por temporada. Fundada em 2018 por egressos da Universidade Federal de Santa Catarina, a empresa teve sua origem em Florianópolis, mas rapidamente expandiu suas operações para diversas outras cidades do Brasil. Desde o início, a Seazone se diferenciou pelo uso proativo de tecnologia para otimizar a gestão de imóveis, oferecendo uma ampla gama de serviços que atendem tanto aos proprietários quanto aos hóspedes.

Inicialmente focada na gestão de imóveis anunciados em plataformas digitais como o Airbnb, a Seazone identificou uma oportunidade significativa de melhorar a apresentação das residências e a gestão das reservas, elementos frequentemente negligenciados por proprietários individuais. Com a Seazone, imóveis que anteriormente eram geridos de forma amadora passaram a ter uma gestão profissional, o que resultou em um aumento substancial do faturamento e na satisfação dos clientes na maioria dos casos.

A Seazone estrutura-se como uma *holding*, tendo duas principais empresas sob seu guarda-chuva: a Seazone Serviços e a Seazone Investimentos. Cada uma delas desempenhando um papel crucial no funcionamento e expansão da empresa.

A Seazone Investimentos tem como papel principal a exploração de oportunidades no mercado imobiliário, focando na identificação de propriedades com alto potencial de rentabilidade no segmento de aluguel por temporada. Esta empresa analisa tendências de mercado e modelos de imóveis, ajudando a atrair investidores e construtoras interessadas em projetos de construção voltados para o aluguel por temporada.

Por sua vez, a Seazone Serviços é responsável pela administração direta das residências. Ela cuida de todos os aspectos da gestão, desde a criação e otimização de anúncios em plataformas como Airbnb e Booking, até a coordenação de tarefas operacionais como check-in, check-out, limpeza e manutenção. É a abordagem profis-

sional da Seazone na administração de imóveis que garante a qualidade dos serviços oferecidos e na rentabilidade das propriedades sob sua gestão.

O contexto do atual projeto de conclusão de curso, se dá dentro do setor de tecnologia da Seazone Serviços. Esta área da empresa é encarregada do desenvolvimento de produtos e sistemas de TI (Tecnologia da Informação) que auxiliam nos processos operacionais, adquirem, refinam e disponibilizam dados para o uso em análises durante a tomada de decisões importantes e promovem a expansão sustentável das demais áreas da empresa.

1.2 MOTIVAÇÃO

Até recentemente, a Seazone recebia a grande maioria do tráfego de aluguel de imóveis através do Airbnb. Esta plataforma foi fundamental para o crescimento inicial da empresa, permitindo uma gestão eficiente e uma otimização contínua dos imóveis listados. No entanto, com o crescimento contínuo da Seazone veio a necessidade de diversificar suas operações, levando a integração de outras plataformas de aluguel por temporada nos processos envolvendo o anúncio de imóveis, em particular, o Booking.

O Booking é uma das maiores e mais conhecidas OTAs (Online Travel Agencies) do mundo, oferecendo uma ampla gama de opções de hospedagem. Enquanto o Airbnb abrange uma grande variedade de tipos de imóveis, com ênfase em propriedades residenciais, o Booking tem um enfoque maior em hotéis, o que amplia as oportunidades de mercado para a Seazone. No momento de escrita da presente monografia, a maior parte dos anúncios da Seazone já foram incluídos na plataforma do Booking e já representam uma porção significativa das reservas da Seazone, o que ressalta a importância desta OTA para a empresa.

Dado a relevância histórica do Airbnb para a Seazone, a empresa já tem a sua disposição um sistema bem estabelecido de aquisição, processamento e análise de dados provenientes dessa plataforma. Este sistema permite que a Seazone obtenha *insights* detalhados sobre preços, disponibilidades e características dos imóveis, utilizando esses dados para otimizar as estratégias de precificação e maximizar a rentabilidade das propriedades. O sistema de dados do Airbnb é um importante diferencial da Seazone, suportando diversas frentes operacionais e estratégicas da empresa.

Com a integração de imóveis da Seazone no Booking, tornou-se essencial desenvolver um sistema similar de aquisição de dados para esta plataforma. Apesar de ambas as OTAs serem parecidas em muitos aspectos, os dados que elas fornecem podem variar significativamente. Por exemplo, algumas propriedades podem estar listadas apenas em uma plataforma e não na outra, e mesmo quando listadas em ambas, as propriedades podem ter preços, disponibilidades e até um perfil de dados completamente diferente do Airbnb, exigindo uma abordagem personalizada para a coleta e análise dessas informações.

A falta de um sistema robusto de aquisição de dados do Booking gera uma lacuna importante na base de dados da Seazone, pois priva setores importantes da empresa de uma gama de informações altamente relevantes para uma variedade de procedimentos. Por exemplo, dados como preço e disponibilidade são fundamentais para a projeção de faturamento de imóveis, que é crucial para que a Seazone Investimentos possa identificar oportunidades de mercado e para a Seazone Serviços ao definir estratégias de precificação com base nos competidores de imóveis Seazone. Informações sobre o número de quartos, localização e avaliação dos anúncios são vitais para identificação de concorrentes diretos e para entender as características dos imóveis em áreas de expansão.

Todos estes fatores apontam para a importância da implementação de um sistema de aquisição de dados para o Booking como um passo estratégico fundamental para a Seazone. Este sistema permitirá à empresa manter-se competitiva, melhorar a precisão das suas análises e apoiar a expansão sustentável das suas operações.

1.3 ORGANIZAÇÃO DO DOCUMENTO

Esta monografia está estruturada da seguinte forma:

O Capítulo 2 apresenta a fundamentação teórica necessária para o entendimento do restante da monografia. São abordados conceitos e tecnologias importantes, contextualizando o projeto dentro do cenário técnico atual.

No Capítulo 3, são descritos os principais problemas abordados por este projeto e os requisitos técnicos que precisam ser cumpridos. Esta seção detalha as limitações e desafios enfrentados na ausência de um sistema robusto de aquisição de dados do Booking e o que tem que ser atendido para aliviar esta situação.

O Capítulo 4 descreve a metodologia de exploração dos sites e a arquitetura geral do sistema de web *scraping*.

O desenvolvimento da solução é detalhado no Capítulo 5, incluindo a implementação dos *scrapers* e a integração com a infraestrutura em nuvem da AWS. São discutidos os passos seguidos, desde a concepção inicial até a automatização completa do sistema.

A análise dos resultados obtidos ao longo do projeto é feita no Capítulo 6. São avaliados os dados coletados, o desempenho dos *scrapers* e a conformidade com os requisitos funcionais e não-funcionais estabelecidos.

Por fim, o Capítulo 7 apresenta as considerações finais, fornecendo um resumo abrangente das atividades e realizações do projeto. São discutidas as contribuições do trabalho para a Seazone, os principais aprendizados e as perspectivas futuras.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo tem como objetivo apresentar as tecnologias e conceitos essenciais para a compreensão dos demais capítulos, além da atual arquitetura do sistema de dados da Seazone. Na Seção 2.1, é contextualizado o site do Booking.com, seguido pela Seção 2.2, que detalha a plataforma Stays. A Seção 2.3 aborda o uso de APIs e, na Seção 2.4, são descritos os formatos de dados relevantes para o projeto. O conceito de web *scraping* é apresentado na Seção 2.5, enquanto a Seção 2.6 introduz o conceito de *data lake*. Na Seção 2.7, é explicado o funcionamento do Docker e, na Seção 2.8, a plataforma de serviços em nuvem Amazon Web Services (AWS). A Seção 2.9 trata da computação concorrente, seguida pela Seção 2.10, que discute Python e bibliotecas relevantes. Por fim, na Seção Seção 2.11, é descrita a arquitetura do data lake da Seazone.

2.1 BOOKING.COM

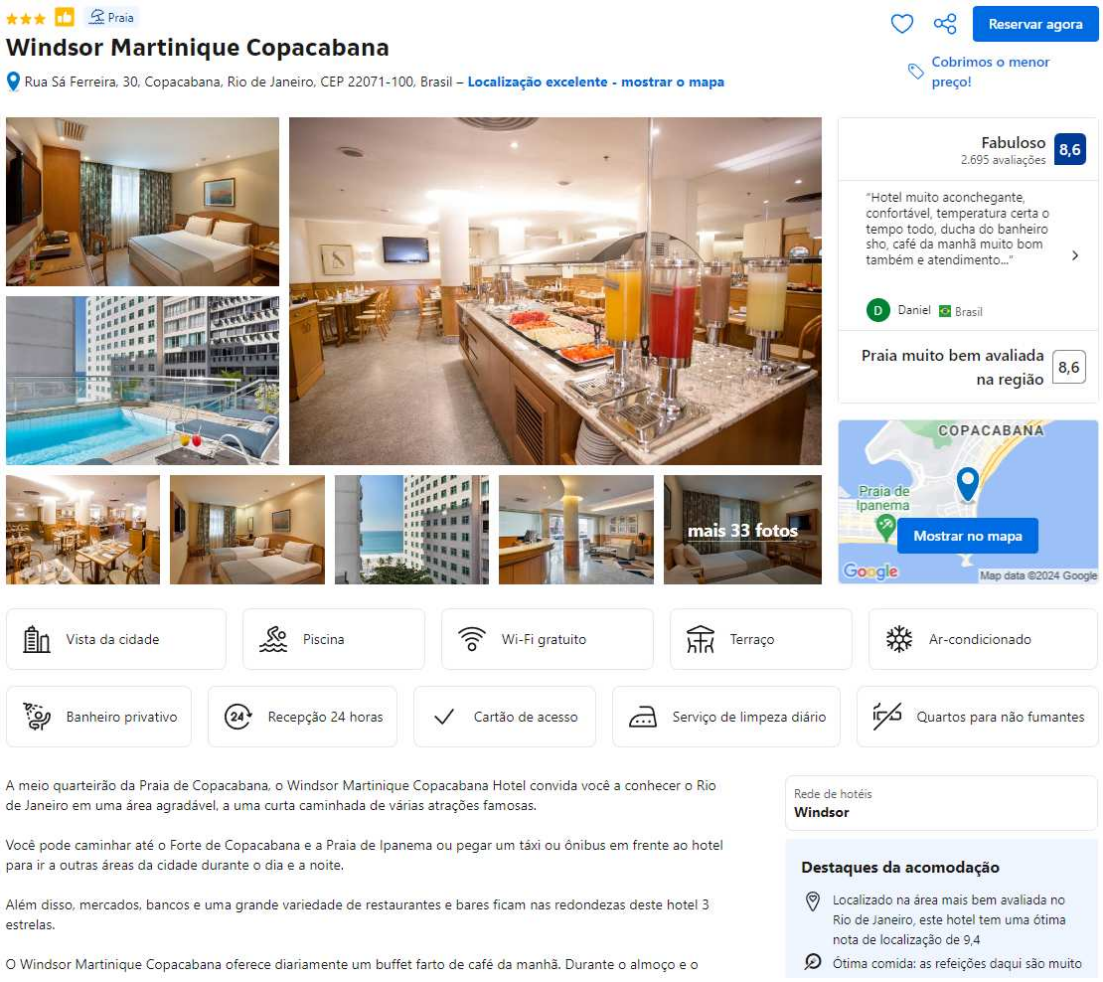
A plataforma Booking.com, fundada em 1996 na Holanda e com sede em Amsterdã, evoluiu de uma pequena *startup* holandesa para uma das maiores empresas de viagens digitais do mundo. Oferecendo uma vasta seleção de hotéis, aluguéis de temporada, hostels, casas de hóspedes, voos, experiências locais, aluguel de carros e outras opções de transporte, a Booking.com se destaca por sua abordagem centrada no usuário, proporcionando uma experiência de reserva eficiente e abrangente (BOOKING.COM, s.d.)

Atualmente, a Booking continua a ser líder no mercado global de viagens, destacando-se pela sua capacidade de atender às necessidades dos viajantes em destinos espalhados pelo mundo inteiro. Boa parte deste sucesso vem do enfoque da empresa na venda de produtos através de sua plataforma online, que atua para facilitar a busca, comparação e reserva de acomodações e outros serviços relacionados à viagens, realizando a maioria das suas vendas através de seu site.

As reservas realizadas no Booking.com são normalmente conduzidas através das páginas específicas de cada anúncio, conforme apresentado na Figura 1, que mostra o exemplo da página de anúncio de um hotel no Booking. Cada página pode conter de um a vários imóveis (ou quartos no caso de hotéis) diferentes e serve como um hub de informações relevantes à estadia, que é parcialmente personalizável pelo anfitrião. O acesso à características como título, descrição, fotos, preços, avaliações, comodidades e muitas outras, são de grande importância para o processo de aluguel de estadia, pois a página do anúncio acaba sendo a principal interface de venda entre anfitrião e cliente. Isso faz da página individual de cada anúncio o local ideal para a aquisição de dados, já que é nela onde estarão concentradas a grande maioria das informações relevantes ao cliente, portanto, a maior parte dos dados de importantes

para as análises da Seazone.

Figura 1 – Captura de tela da página de um hotel no Booking.com.



Fonte: Elaborado pelo autor.

2.2 STAYS

A Stays é uma plataforma de gerenciamento de imóveis voltada para a administração de propriedades em diversas OTAs. Sua funcionalidade central é permitir que proprietários e gestores de imóveis listem e gerenciem suas propriedades simultaneamente em múltiplas plataformas de aluguel por temporada, sem enfrentar problemas de conflitos ou inconsistências, tendo integração nativa com o Airbnb e o Booking.

Para a Seazone, a Stays desempenha um papel crucial na integração e centralização das operações de gerenciamento de imóveis. Utilizando a Stays, a Seazone consegue sincronizar automaticamente as informações de disponibilidade, preços e reservas de seus imóveis em todas as OTAs, garantindo que não haja duplicidade de reservas ou discrepâncias entre anúncios. Essa integração facilita a otimização das

listagens, melhora a eficiência operacional e aumenta a visibilidade dos imóveis, permitindo uma gestão mais eficaz, e, por consequência, aumentando a taxa de ocupação dos imóveis.

2.3 APIS

As APIs (Interfaces de Programação de Aplicações) permitem que diferentes programas de software se comuniquem entre si. Elas são projetadas para abstrair a complexidade dos sistemas, expondo apenas as funcionalidades necessárias para os desenvolvedores. Isso facilita a integração entre aplicações e permite que os desenvolvedores aproveitem as funcionalidades de um serviço sem precisar entender todos os detalhes de como ele opera internamente.

O funcionamento das APIs se da através de chamadas ou requisições feitas de um cliente para um servidor, que responde com os dados ou ações solicitadas. Essas chamadas são geralmente feitas usando protocolos de comunicação como HTTP (Hypertext Transfer Protocol), REST (Representational State Transfer) e SOAP (Simple Object Access Protocol), e podem retornar dados em vários formatos, sendo JSON (JavaScript Object Notation) e XML (Extensible Markup Language) dois dos mais comuns.

Por conta da sua praticidade, as APIs acabam desempenhando um papel crucial no desenvolvimento de software moderno, estando presentes em praticamente todas as aplicações web modernas. Por exemplo, quando um usuário visualiza um mapa integrado no site, é provável que esse mapa esteja sendo fornecido por uma API de um serviço de mapas. Da mesma forma, sistemas de pagamento online, redes sociais, e serviços de *streaming* de música e vídeo, frequentemente utilizam APIs para fornecer os seus serviços.

Isso tudo acaba sendo possível pela compartimentalização proporcionada pelas APIs, que costumam ser projetadas de forma individual para desempenharem tarefas específicas independentes de um sistema maior. Esse desacoplamento acaba abrindo espaço para o desenvolvimento e a comercialização de serviços e funcionalidades específicas através da disponibilização de APIs. Empresas podem criar APIs para serviços, como processamento de pagamentos, verificação de identidade, análise de dados, entre outros, e vendê-las a outras empresas que necessitem dessas funcionalidades.

2.4 FORMATOS DE DADOS RELEVANTES

Para uma boa compreensão desta monografia, é relevante o conhecimento de alguns formatos de dados que são utilizados durante os processos de aquisição e armazenamento de informações provenientes da plataforma Booking. Os três principais

formatos de dados abordados neste contexto são JSON, HTML e Parquet. Cada um desses formatos possui características específicas que os tornam adequados para diferentes etapas do projeto. A seguir, será apresentada a descrição de cada um desses formatos de dados.

2.4.1 JSON

JSON é um formato de troca de dados leve e de fácil leitura/escrita tanto por humanos quanto por máquinas. Conforme destacado em (SEVERANCE, 2012), JSON surgiu como um padrão amplamente adotado para a transmissão de dados na web devido à sua simplicidade de serialização e desserialização, facilitando a interoperabilidade entre sistemas distintos. Esse formato de dados foi originalmente concebido no JavaScript, contudo, ele é amplamente utilizado de forma independente em diversas linguagens de programação da atualidade.

Um dos principais benefícios do JSON é sua estrutura de chave-valor, que facilita a organização e o acesso aos dados. Este formato é particularmente útil para representar dados estruturados e semi-estruturados, permitindo o aninhamento de objetos e listas. A sintaxe simples e a compatibilidade com várias linguagens de programação fazem do JSON uma escolha popular para APIs.

2.4.2 HTML

HTML é uma das linguagens de marcação mais utilizadas no desenvolvimento de páginas web. Ela consiste de uma série de elementos que definem o conteúdo e a estrutura de um documento web, com cada elemento HTML podendo incluir atributos que especificam características adicionais, como classes, IDs, e estilos. O HTML permite a incorporação de texto, imagens, links e outros tipos de mídia em uma página web, por isso é a base de boa parte das páginas na internet, sendo essencial para a apresentação e navegação de conteúdo online.

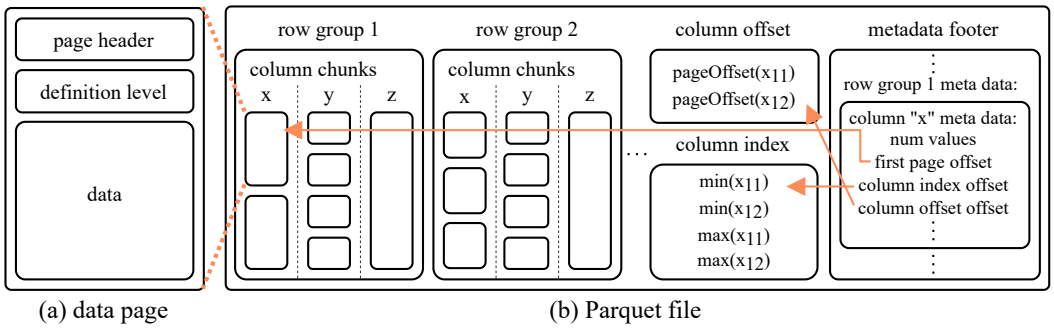
2.4.3 Parquet

Parquet é um formato colunar semi-estruturado de dados, focado em desempenho. Desenvolvido originalmente pelo Twitter e pelo Cloudera, o Parquet é especialmente adequado para o uso em sistemas de análise de dados distribuídos, como Hadoop e Spark. Diferente do modelo tabular tradicional, que organiza dados em valores x atributos, o Parquet organiza as informações em atributos x valores, trazendo um série de benefícios do ponto de vista de performance de máquina, em troca de uma formatação com baixa legibilidade humana.

Este tipo de arquivo é utilizado primariamente em casos onde o tamanho e velocidade de leitura dos objetos é crítico. Devido à sua alta compressibilidade, ele

frequentemente ocupa menos espaço que outros formatos mais usuais de dados, como o JSON e o CSV por exemplo. Conforme evidenciado em (ALICE; MICHAEL; THOMAS, 2023), a estrutura inteligente do Parquet (esboçada na Figura 2), providencia um severo aumento no desempenho de leitura dos arquivos, o que se dá primariamente pela inclusão de metadados relevantes em sua esquemática e pelo formato colunar separado em grupos indexados de dados.

Figura 2 – Estrutura de arquivo Parquet.



Fonte: (ALICE; MICHAEL; THOMAS, 2023).

2.5 WEB SCRAPING

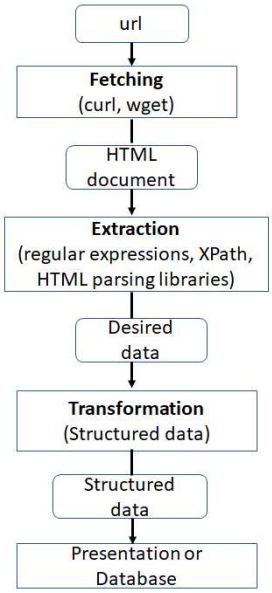
Web *scraping* é a técnica de extração automática de dados em páginas da web utilizando um software especializado (KHDER, 2021). Esta técnica permite a transformação de dados não estruturados, em informação relevante e estruturada, que pode ser armazenada e analisada posteriormente (SIRISURIYA, 2015). A prática de web *scraping* já é bem estabelecida em diversos setores da indústria, desempenhando um papel crucial na aquisição de dados para algorítimos de inteligência artificial e análises de inteligência de negócios, além de servir como principal mecanismo de navegação e indexação de páginas na web em motores de busca como o Google, Bing e Yahoo.

2.5.1 Etapas de *scrapagem*

Segundo (KHDER, 2021), o processo de *web scraping* (também referenciado como *scrapagem* ao longo desta monografia) pode ser generalizado em três etapas esquematizadas no fluxograma da Figura 3 e descritas a seguir:

1. **Etapa de busca:** Aqui o local de interesse onde se encontram as informações que serão extraídas é acessado pelo código, o que é tipicamente feito através de requisições ao endereço alvo. Muitas vezes é necessário realizar essa etapa

Figura 3 – Fluxograma do processo generalizado de scrapagem.



Fonte: (KHDER, 2021).

de maneira à emular o comportamento de um usuário padrão fazendo o acesso pelo navegador para que o site aceite a chamada sem bloqueá-la.

2. **Etapa de extração:** Após obter o resultado crú da requisição (usualmente retornado em formato HTML, JSON ou XML), é feito a extração das informações de interesse. O procedimento de extração varia de acordo com a formatação dos dados retornados, mas costuma ser realizada através de ferramentas específicas de análise sintática de dados. Alguns exemplos de tecnologias utilizadas nessa etapa são: *regex* para *strings* generalizadas, XPath para dados em XML e bibliotecas especializadas em navegação de HTML como o Jsoup para JavaScript e BeatifulSoup para Python.

3. **Etapa de transformação:** Por fim, é feita a transformação dos dados relevantes para algum formato estruturado, visando a apresentação ou armazenamento deles. Esta etapa é onde o *scraper* (*script* que realiza o *web scraping*) tipicamente faz a conexão direta ou indireta com o mecanismo de armazenamento de dados (bancos de dados, planilhas, *data lakes*, etc.), que serão usados posteriormente para qualquer que seja a finalidade das informações adquiridas.

2.5.2 Métodos relevantes de scrapagem

Conforme exposto na subseção anterior, o *web scraping* pode ser feito de diferentes formas a depender de aspectos como a natureza do site onde os dados

se encontram, linguagem de programação de preferência e mecanismo de armazenamento de dados disponível. Posto isso, dois métodos de scrapagem se destacam como especialmente relevantes para este PFC: scrapagem por HTML de página estática e por uso de APIs públicas do site.

A extração de dados diretamente do HTML, é um método tradicional e amplamente utilizado em web*scraping*. Esse método envolve o download e análise do HTML da página web, a fim de localizar e extrair os dados desejados. A etapa de busca deste método costuma ser feita através de uma requisição HTTP ao URL (Uniform Resource Locator) padrão da página (o URL na qual o usuário padrão utilizaria para acessar o site). O HTML, sendo uma linguagem de marcação, organiza os dados em uma estrutura hierárquica que pode ser navegada usando várias bibliotecas e ferramentas de programação na etapa de extração.

Este método é particularmente útil quando os dados são apresentados de maneira consistente e previsível dentro da estrutura HTML da página. A desvantagem, no entanto, é que qualquer alteração no *layout* da página pelo proprietário do site pode quebrar o *scraper*, exigindo ajustes no código. Além disso, este método também tende a ser mais lento que o método de scrapagem por API, pois a a navegação em HTML acaba sendo relativamente custosa em termos de processamento à depender do tamanho da página.

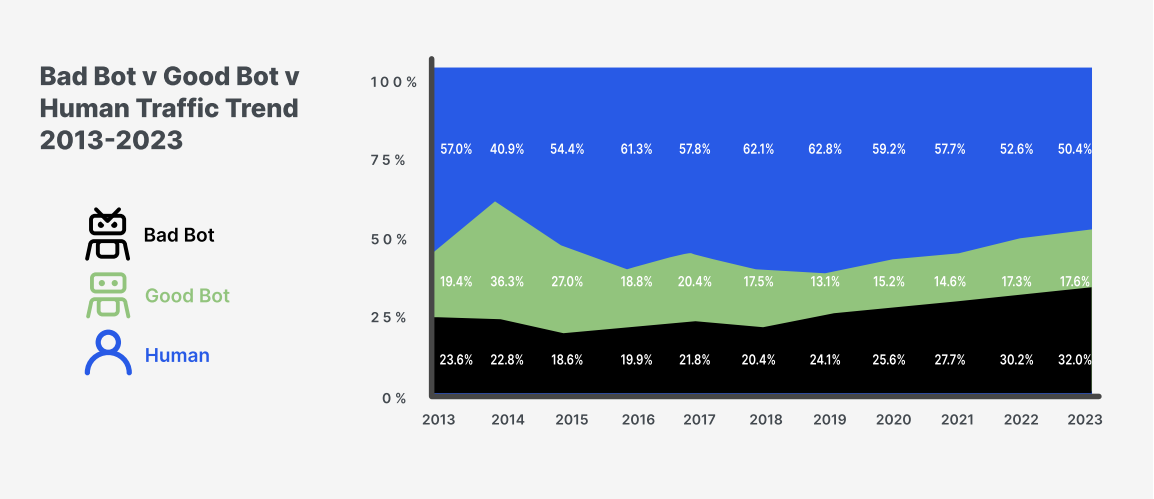
Outro método eficaz de scrapagem, é a utilização das APIs públicas do site. Aqui a etapa de busca é realizada por requisições à URL específica da API. O endereço e método de chamada das APIs, nem sempre são repassados explicitamente, nesses casos é necessário a aplicação de engenharia reversa nas chamadas feitas ao *back-end* pelo navegador, assim descobrindo a maneira correta de fazer as requisições para obter os dados de interesse. Na maioria dos casos, os dados retornados pelas chamadas de API deste método estão no formato JSON, portanto a etapa de extração costuma feita através de interpretadores de JSON.

A aplicação deste método só é possível em casos onde existem APIs que retornam as informações de interesse para a scrapagem, o que nem sempre é o caso, já que não é necessário o uso de APIs para uma infinidade de processos da web. Apesar disso, este método acaba sendo a escolha mais atraente para quase todo caso onde ele é aplicável, o que se da principalmente pelo fato dos dados retornados pelas APIs internas serem geralmente formatados em JSON, que é facilmente navegável pela maioria das linguagens de programação utilizadas em web *scraping*, tornando o processo de extração muito menos custoso que o de HTML. Além disso, as informações retornadas por APIs costumam apresentar padrões muito mais estáveis, resultando em uma menor taxa de manutenção de código quando comparado com o outro método abordado.

2.5.3 Empecilhos de scrapagem

Boa parte do tráfego da internet é proveniente de robôs, que neste contexto representam qualquer software de acesso automático a páginas da web. De acordo com um estudo realizado em 2024 pela Imperva (IMPERVA, 2024), a proporção de acessos na web realizados em 2023 por robôs é de 49,6%, número que vem crescendo nos últimos anos, conforme ilustrado na Figura 4. Esses robôs podem ser separados em dois grupos: robôs bons e robôs maus. Robôs bons são aqueles que provem qualquer tipo de funcionalidade de valor para os sites que acessam, um exemplo disso sendo os Googlebots, que indexam as páginas da web para o ranqueamento de sites no motor de busca do Google (IMPERVA, 2024). Qualquer robô não categorizado como bom, pode ser considerado mau, isso inclui desde robôs utilizados em ataques de DDoS (*denial-of-service*) até os próprios web *scrapers*.

Figura 4 – Gráfico da proporção de trafego na internet entre humanos, robôs bons e robôs maus de 2013 a 2023.



Fonte: (IMPERVA, 2024).

Os web sites geralmente não apreciam qualquer tipo de atividade proveniente de robôs maus em suas plataformas, pois podem representar ameaças à integridade do site ou, no melhor dos casos, trafego artificial que não será convertido em possíveis clientes. Para mitigar essas questões, os web sites implementam diversas medidas de segurança e restrições para dificultar a ação desses robôs, como CAPTCHAs, que são basicamente mecanismos de detecção por análise de padrões de navegação, limitação de velocidade de acesso e verificações de elementos HTTP para identificar e bloquear requisições indesejadas.

A presença de tais impedimentos apresenta desafios significativos para os *scrapers*, especialmente para casos como o da Seazone, em que há a necessidade de

coletar grandes volumes de dados para análises detalhadas e alimentações contínuas de sistemas de dados. Os *web scrapers* adotam diversas estratégias para contornar esses desafios. Uma abordagem comum é a de simular o comportamento de um usuário humano, alterando cabeçalhos HTTP, usando diferentes *User-Agents* (identificadores de software em requisições web) e implementando atrasos aleatórios entre as requisições para evitar a detecção. Em casos mais extremos de bloqueio, também pode ser feito o uso de *proxies* para distribuir as requisições através de múltiplos IPs ou até a redução forçada da velocidade de scrapagem.

2.6 DATA LAKE

O *data lake* consiste de um repositório de armazenamento altamente escalonável que contém vastas quantidades de dados brutos após terem passado por um processo de formatação mínimo ou nulo, e de um sistema de processamento capaz de inserir dados sem comprometer a estrutura do repositório (LASKOWSKI, 2016). *Data lakes* são tipicamente construídos para suportar a ingestão de largas quantidades de dados heterogêneos, que são armazenados de forma à estarem disponíveis para o uso a partir momento em que chegam no repositório (MILOSLAVSKAYA; TOLSTOY, 2016).

Ao contrário de sistemas de gerenciamento de bancos de dados tradicionais, os *data lakes* usam uma arquitetura plana de armazenamento, tipicamente compartimentalizando os dados em objetos (STEDMAN; LUTKEVICH, s.d.). Os objetos dentro de um *data lake* costumam ser armazenados em formatos especializados arquivo, com alta compressibilidade e velocidade de leitura, como o Parquet, ORC e Avro. Para uma maior usabilidade dos dados, os *data lakes* são normalmente auxiliados por uma camada semântica que da contexto e define a estrutura e relação entre os dados do repositório, assim permitindo o uso de ferramentas como o SQL (Structured Query Language) durante a consulta e processamento de informações (MILOSLAVSKAYA; TOLSTOY, 2016).

2.7 DOCKER

Docker é uma plataforma de código aberto que executa aplicações e torna mais fácil o processo de desenvolvimento e distribuição de software. Aplicações construídas em Docker, são empacotadas com todas as dependências de suporte num formato padrão chamado de contêiner (BASHARI; JOHN; MOHAMMAD, 2017).

Os contêiner permitem que múltiplos componentes diferentes de uma mesma aplicação possam compartilhar dos recursos de uma único sistema operacional, sem permitir que elas acessem ou visualizem umas às outras, o que é realizado através

do aproveitamento das funcionalidades de virtualização e isolamento de processos proporcionadas pelo *kernel* do sistema operacional Linux (IBM, s.d.).

O conjunto de instruções necessárias para criar um contêiner Docker é chamado de imagem Docker. Essencialmente, a imagem é um *template* estático a partir do qual os contêineres são iniciados, contendo todos os aspectos necessários para a execução do contêiner, como o código fonte, dependências, variáveis de ambiente e bibliotecas (IBM, s.d.). As imagens Docker são compostas por camadas que representam diferentes instruções no Dockerfile, que é o *script* com as definições de como a imagem deve ser construída.

A capacidade de executar aplicações em um ambiente praticamente idêntico ao ambiente de origem, independente de circunstâncias externas ao programa, como o hardware e sistema operacional da máquina onde a aplicação está sendo executada, aliado à praticidade proporcionada pelo uso de imagens Docker, faz do Docker uma ferramenta de desenvolvimento de software poderosíssima e altamente disseminada na indústria moderna de TI, em que boa parte das aplicações precisam ser executadas em nuvem, tornando mecanismos de conteinerização algo muito valioso.

2.8 AMAZON WEB SERVICES

Lançada em 2006, a AWS é a plataforma de nuvem mais adotada do mundo, oferecendo mais de 200 serviços completos e *datacenters* em todo o mundo (AMAZON WEB SERVICES, s.d.[j]). Boa parte do sucesso e popularidade da plataforma vem do ecossistema de serviços proporcionados pela filial da Amazon, que oferecem abstrações poderosas de sistemas e processos que frequentemente aparecem em soluções de TI, além de proporcionar ao cliente a infraestrutura necessária para suportar demandas de praticamente qualquer magnitude. É por essas vantagens, aliadas à cobrança flexível disponibilizada pela maioria dos seus serviços, que a Seazone tem a AWS como plataforma de nuvem primária da empresa.

Quase todas as aplicações do setor de dados da Seazone usufruem de um ou mais serviços AWS em seu funcionamento. Nesta seção, será introduzido alguns dos serviços com maior relevância para o atual PFC.

2.8.1 S3

Amazon S3 (Simple Storage Service) é um serviço de armazenamento de objetos projetado para armazenar e recuperar qualquer quantidade de dados a qualquer momento, de qualquer lugar na web (AMAZON WEB SERVICES, s.d.[f]). Ele fornece uma infraestrutura altamente escalável, durável e segura para armazenar dados em *buckets*, que são recipientes para deposito de objetos como arquivos, imagens, vídeos e backups de dados, em uma arquitetura plana de armazenamento.

O S3 oferece uma série de funcionalidades, como controle de acesso, versionamento de objetos, e a capacidade de definir políticas de ciclo de vida para gerenciar os dados armazenados, além de possuir integração nativa com vários outros serviços da AWS, permitindo o processamento e análise de dados diretamente no ambiente de armazenamento (AMAZON WEB SERVICES, s.d.[f]). O S3 tem custos de armazenamento relativamente baixos quando comparado com sistemas mais tradicionais, sendo a cobrança do S3 proporcional ao tamanho total dos objetos armazenados. Esses fatores fazem do S3 um serviço altamente relevante em aplicações de *big data*, com ele sendo usado como o principal repositório de armazenamento de dados no *data lake* da Seazone.

Uma funcionalidade especialmente relevante do S3 é o particionamento de objetos. Mesmo com uma hierarquia de armazenamento plana, o S3 permite organizar objetos em diretórios lógicos através de prefixos nos nomes dos arquivos. O particionamento de objetos permite que grandes conjuntos de dados sejam divididos em segmentos menores, otimizando o desempenho e a velocidade das operações de leitura e escrita, essencial para aplicações que exigem alta performance e rapidez, como um *data lake*.

2.8.2 ECS

Amazon Elastic Container Service (ECS) é um serviço de orquestração de contêineres altamente escalável e de alta performance, permitindo que aplicações sejam facilmente executadas e gerenciadas na AWS (AMAZON WEB SERVICES, s.d.[e]). O ECS facilita a implantação, o gerenciamento e a escalabilidade de contêineres em *clusters* de máquinas virtuais, suportando diferentes tipos de arquiteturas, desde simples aplicações monolíticas até aplicações de micro-serviços complexas. Isso é feito primariamente através das tarefas ECS, que são as unidades básicas de trabalho no ECS, contendo as definições de como os contêineres devem ser executados, as especificações dos recursos necessários, como CPU (Central Processing Unit) e RAM (Random Access Memory) da instância, e as configurações de rede.

Existem dois tipos de execução de tarefas no ECS, porém o único tipo relevante para este projeto é o AWS Fargate, que consiste em uma execução sem a necessidade de gerenciar servidores ou *clusters* de máquinas. Com o Fargate, as instâncias da aplicação desejada são provisionadas e escalonadas de maneira automatizada através de configurações mínimas de tarefa. Por conta da flexibilidade proporcionada por esse serviço, todos os scrapers da Seazone são executados em contêineres Docker gerenciados por Fargate.

Também existe a funcionalidade de serviços ECS, que permitem a definição e manutenção da forma de execução desejada para um determinado grupo de tarefas. Um serviço ECS gerencia automaticamente a distribuição e o balanceamento de carga

de contêineres em um *cluster* especificado, de forma à garantir que a quantidade necessária de tarefas seja executada continuamente, reiniciando-as se elas falharem ou forem encerradas. Essa funcionalidade é crucial para manter a alta disponibilidade e resiliência de certas aplicações, além de facilitar a escalabilidade e o gerenciamento automatizado de processos executados em ECS.

2.8.3 ECR

O Amazon Elastic Container Registry (ECR) é um serviço de registro de contêineres que facilita o armazenamento, gerenciamento e implantação de imagens de contêineres (AMAZON WEB SERVICES, s.d.[d]). Ele possui integração nativa com a maioria dos serviços de computação da AWS, com o ECS sendo um deles. Todas as tarefas ECS Fargate de *scrapers* puxam a imagem do contêiner que usam, diretamente do ECR.

2.8.4 VPC

Uma VPC (Virtual Private Cloud) é uma rede virtual privada dentro da AWS que permite o provisionamento de uma seção isolada da nuvem, onde recursos podem ser executados em um ambiente controlado (AMAZON WEB SERVICES, s.d.[h]). As VPCs são compostas por *subnets*, que são sub-redes divididas em áreas públicas e privadas, facilitando a organização e a segurança dos recursos. Em aplicações executadas em ECS (Elastic Container Service), as VPCs são utilizadas para definir o ambiente de rede no qual os contêineres operam, garantindo controle sobre o tráfego de rede e segurança. As VPCs serão usadas como interface de rede para a comunicação dos *scripts* de aquisição de dados da Seazone com a internet, disponibilizando um amontoado de IPs públicos para os *scrapers*.

2.8.5 EC2

Amazon EC2 (Elastic Compute Cloud) é um serviço da AWS que fornece capacidade de computação escalável na nuvem (AMAZON WEB SERVICES, s.d.[c]). Ele permite os usuários à lançar, configurar e gerenciar instâncias de servidores virtuais de acordo com suas necessidades. O EC2 oferece uma variedade de tipos de instâncias, otimizadas para diferentes casos de uso, como computação de alto desempenho, armazenamento em disco SSD e memória otimizada. No contexto do PFC, este serviço é usado primariamente em testes pontuais de web *scrapers* para otimizar o desempenho deles em nuvem.

2.8.6 SQS

Amazon Simple Queue Service (SQS) é um serviço de filas de mensagens totalmente gerenciado que permite o desacoplamento e a escalabilidade de microsserviços, sistemas distribuídos e aplicações *serverless* (modelo de computação sem a necessidade de provisionamento ou gerenciamento de servidores) (AMAZON WEB SERVICES, s.d.[g]). O SQS oferece dois tipos de filas: *standard* e FIFO (First-In-First-Out). As filas FIFO garantem que as mensagens sejam processadas exatamente uma vez, na ordem em que foram enviadas, sendo ideais para situações onde a ordem e/ou a singularidade de processamento são importantes. Filas SQS do tipo FIFO são utilizadas em quase todo *scraper* para que possam funcionar de forma distribuída sem que *scrapem* uma mensagem mais de uma vez (filas *standard* podem acabar entregando uma mesma mensagem múltiplas vezes).

2.8.7 Amazon Kinesis Data Firehose

Amazon Kinesis Data Firehose é um serviço gerenciado de entrega em tempo real de dados de *streaming* para uma variedade de destinos dentro e fora da AWS. Ele permite a captura, processamento e carregamento de dados com relativa facilidade, garantindo alta escalabilidade e confiabilidade sem a necessidade de gerenciar infraestrutura. O Firehose também oferece suporte a transformações personalizadas de dados, permitindo a aplicação de processos como mudança de formato e particionamento de dados. Este serviço é usado para fazer a transformação dos dados adquiridos pelos *web scrapers* e a ingestão deles no *data lake*.

2.8.8 Amazon Athena

Amazon Athena é um serviço *serverless* de consulta interativa que permite analisar dados diretamente no Amazon S3 usando o SQL (AMAZON WEB SERVICES, s.d.[b]). As consultas são executadas em paralelo, proporcionando rápidos resultados mesmo em casos envolvendo um grande volume de dados, com a cobrança do serviço sendo proporcional à quantidade de dados escaneados pela *engine*. O Athena pode processar diversos formatos de dados, como CSV, JSON, ORC, Avro e Parquet, tornando-o ideal para o uso em sistemas de *data lake*. O Amazon Athena é o principal mecanismo de consulta de dados no *data lake* da Seazone, sendo utilizado tanto em análises pontuais de dados, quanto em alguns processos mais complexos de ETL (Extração, Transformação e Carregamento).

2.8.9 AWS Glue

AWS Glue é um serviço de integração de dados *serverless* que facilita a preparação e o carregamento de dados para análise. Ele oferece suporte para ETL, permitindo

que os usuários cataloguem, limpem, enriqueçam e movam dados entre vários repositórios, sem a necessidade de gerenciamento de infraestrutura, simplificando uma série de processos essenciais (AMAZON WEB SERVICES, s.d.[i]).

O AWS Glue é composto de diversos subserviços com funcionalidades diferentes, porém complementares entre si. O catálogo de dados Glue é um repositório central de metadados onde são armazenadas informações sobre os dados da conta, atuando como um índice que descreve a estrutura e localização dos dados armazenados em diversos repositórios, facilitando a descoberta e gerenciamento dos dados para operações de ETL. As tabelas Glue fazem parte dos catálogos de dados Glue e representam uma estrutura de dados específica, como uma tabela em um banco de dados relacional. Os *crawlers* do Glue são componentes que exploram fontes de dados de interesse, inferindo esquemas e criando metadados automaticamente no catálogo do Glue.

O catálogo de dados Glue e as tabelas Glue são fundamentais para o funcionamento do Athena na infraestrutura de dados Seazone, proporcionando a esquematização necessária para que as informações no *data lake* sejam legíveis através do Athena. Os *crawlers* servem para fazer a criação inicial e atualização periódica dos metadados e tabelas Glue.

2.8.10 Step Functions

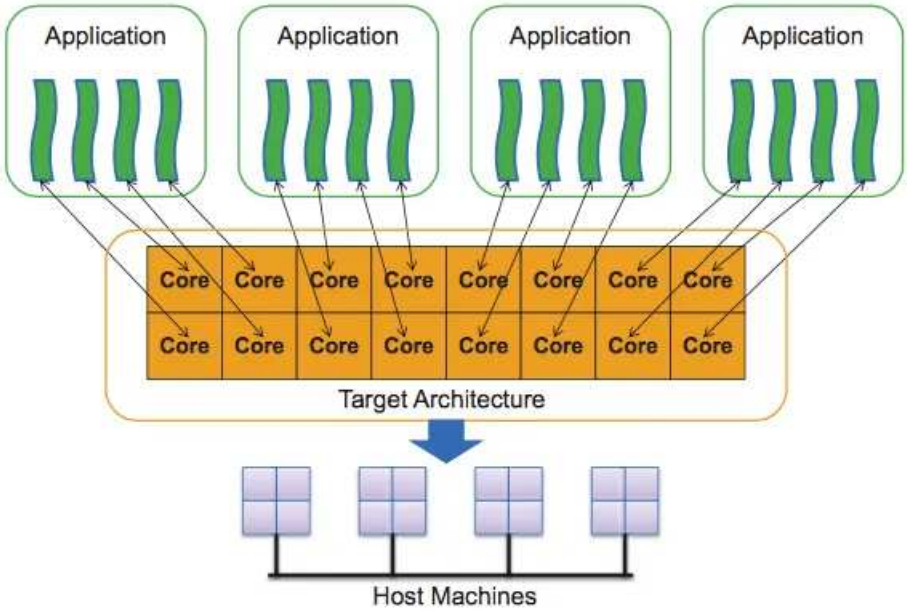
AWS Step Functions é um serviço de orquestração de componentes distribuídos e microsserviços em aplicações. Utilizando fluxos de trabalho visuais definidos por máquinas de estado, é possível definir e monitorar sequências de atividades ou tarefas em nuvem, garantindo o controle sobre a execução de processos complexos. O Step Functions possui integração nativa com praticamente todos os serviços AWS, facilitando bastante o processo de orquestração no ecossistema AWS. Atualmente a maioria dos scrapers da Seazone são agendados e coordenados através do MWAA (Managed Workflows for Apache Airflow), que é outro serviço de orquestração de tarefas, contudo, os *scrapers* implementados ao decorrer deste projeto serão orquestrados no Step Functions, pois ele oferece um nível de integração com serviços AWS superior ao MWAA a um preço consideravelmente reduzido.

2.9 COMPUTAÇÃO CONCORRENTE

A computação concorrente é um tipo de computação em que múltiplos cálculos ou processos são efetuados simultaneamente, o que pode ser feito através da utilização de vários processadores, núcleos ou computadores (SRUJAN, 2023), conforme ilustrado na Figura 5. Quando aplicado corretamente, este método pode melhorar a eficiência e o desempenho das aplicações, sendo essa uma abordagem essencial em

sistemas modernos, onde múltiplos processos e *threads* (fluxos de controle para a execução de processos) podem ser executados paralelamente, aproveitando ao máximo o poder de processamento disponível.

Figura 5 – Representação de várias aplicações executadas simultaneamente através de múltiplos processos, núcleos e computadores.



Fonte: (SRUJAN, 2023).

2.9.1 Tarefas I/O-bound e CPU-bound

Existem dois tipos principais de tarefas na computação concorrente: tarefas *I/O-bound* e tarefas *CPU-bound*. Tarefas *I/O-bound* são aquelas que dependem principalmente de operações de entrada/saída, como ler e escrever em discos ou redes. Essas tarefas geralmente ficam aguardando a conclusão de operações de I/O e a concorrência permite que outras tarefas sejam executadas enquanto isso ocorre, sem a necessidade do uso de múltiplos núcleos de CPU. Já as tarefas *CPU-bound* são intensivas em processamento e dependem do poder de processamento do CPU. Para que essas tarefas fiquem mais rápidas, é necessário dividir a carga de trabalho entre múltiplas unidades de processamento.

2.9.2 Paralelismo de dados

O paralelismo de dados envolve a divisão de um grande conjunto de dados em partes menores, permitindo que essas partes sejam processadas simultaneamente

usando a mesma operação. Esse tipo de paralelismo é especialmente eficaz quando uma mesma operação precisa ser executada em múltiplos elementos de dados (SRUJAN, 2023).

Uma das arquiteturas chave usadas para o paralelismo de dados é a SIMD (Single Instruction, Multiple Data). Na abordagem SIMD, uma única instrução é executada simultaneamente em múltiplos elementos de dados. Isso é alcançado por meio de uma unidade de controle que transmite a mesma instrução para várias unidades de processamento, cada uma operando em seu respectivo elemento de dados, processo que é repetido até que a tarefa em questão seja concluída. Arquiteturas SIMD são frequentemente encontradas em processadores vetoriais, GPUs e conjuntos de instruções multimídia (SRUJAN, 2023).

2.10 PYTHON E BIBLIOTECAS RELEVANTES

Python é a linguagem de programação mais usada na área de ciência e engenharia de dados devido à sua sintaxe clara e legível, além de uma vasta coleção de bibliotecas especializadas que facilitam a manipulação, análise e visualização de dados. Sua versatilidade e facilidade de integração com outros sistemas tornam-a uma escolha popular para projetos de dados.

No contexto deste PFC, o Python cumprirá o papel de linguagem de programação primária no desenvolvimento de todo e qualquer scraper e/ou *script* adjacente. Visando a compreensão do funcionamento das principais etapas do projeto, na sequência abaixo, seguem as descrições básicas das bibliotecas de maior relevância para o PFC:

- **BeatifulSoup:** Biblioteca utilizada para extrair dados de arquivos HTML e XML. Ela faz isso através da criação de uma árvore de análise, permitindo a navegação pelo código HTML e a extração de dados de maneira eficiente (RICHARDSON, s.d.). O BeatifulSoup funciona de maneira muito intuitiva e tem compatibilidade com a maioria dos interpretadores sintáticos de HTML e XML, fazendo dessa uma das bibliotecas mais populares para aplicações de web *scraping*;

- **Requests:** Realiza o envio de requisições HTTP de maneira simples e elegante. Ela abstrai a complexidade envolvida no envio de requisições, manipulação de *cookies* e manutenção de sessões, tornando o processo de comunicação com páginas web mais direto, porém mantendo certo grau de personalização de chamadas;

- **Boto3:** SDK (Software Development Kit) oficial da AWS para Python, permitindo a interação com quase qualquer serviço da AWS. O Boto3 disponibiliza uma coletânea de objetos em Python que facilitam a comunicação com serviços da nuvem

por meio de chamadas de API. Com ele, é possível criar, configurar e gerenciar os recursos da plataforma diretamente no código (AMAZON WEB SERVICES, s.d.[a]), facilitando a automação e integração com a infraestrutura da AWS;

- **Eventlet:** Biblioteca especializada em programação concorrente que utiliza o conceito de *green threads*, que são *threads* gerenciadas em nível de usuário ao invés de serem gerenciadas pelo sistema operacional, realizando a execução concorrente de várias tarefas dentro de um único processo através de técnicas de cooperação entre tarefas. O Eventlet é especialmente útil em aplicações que requerem alta escalabilidade, principalemente em processos categorizados como *I/O-bound*, pois a concorrência do Eventlet não é capaz de utilizar núcleos diferentes de CPU;

- **Fire:** Ferramenta de código aberto desenvolvida pelo Google que facilita a criação de interfaces de linha de comando (CLI) a partir de qualquer código Python. Ao usar a Fire, é possível transformar rapidamente funções, classes, objetos, e módulos, em comandos CLI, tornando a execução e teste de *scripts* mais intuitiva e acessível. Essa biblioteca é especialmente útil para desenvolvedores que desejam criar CLIs robustas sem a necessidade de escrever código adicional complexo, permitindo uma integração eficiente e ágil no fluxo de desenvolvimento.

- **Sqs.py:** É uma biblioteca interna da Seazone utilizada para abstrair a interação entre *scrapers* e filas SQS. Ela é responsável por paralelizar o consumo de mensagens SQS essenciais para o funcionamento dos *scrapers* (URLs de sites a serem scrapados, por exemplo) através do uso das *green threads* do Eventlet. Ela também cuida da lógica de retentativa de mensagens, adicionando robustez ao sistema de *web scraping*. O funcionamento e utilidade desta biblioteca serão melhor evidenciados no Capítulo 5.

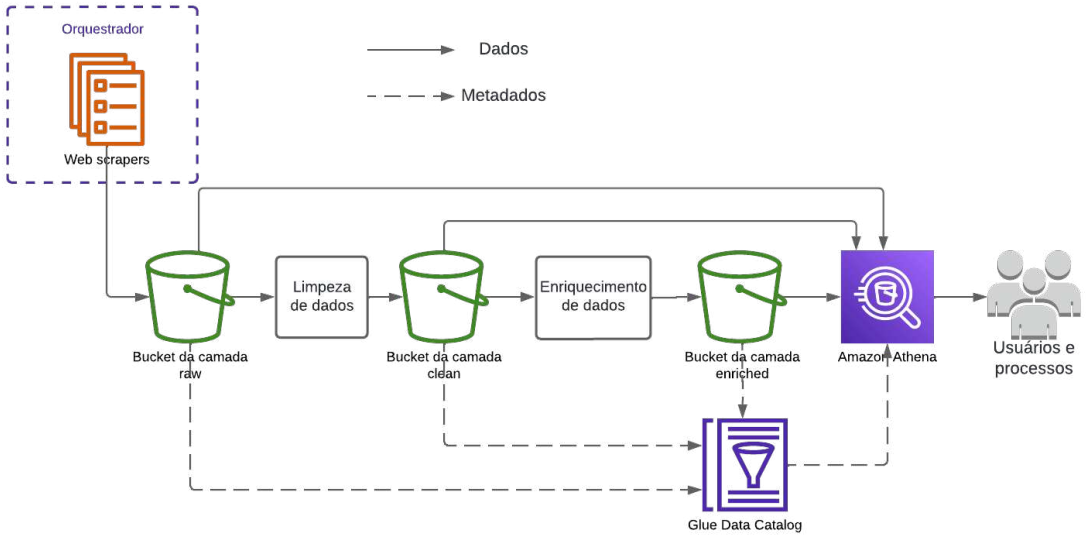
2.11 ARQUITETURA DO DATA LAKE DA SEAZONE

A seguir, será descrito a estrutura geral do *data lake*, segmentando suas funcionalidades em três subseções: armazenamento, aquisição e ingestão, e entrega de dados. A Figura 6 apresenta a arquitetura generalizada do *data lake* da Seazone por meio de um fluxograma, abstraindo algumas partes de baixa relevância para o projeto.

2.11.1 Armazenamento de dados

Os dados do *data lake* são armazenados em *bukets* S3 no formato Parquet. Conforme explicado na seção 2.8, o S3 providencia repositórios de objetos com arquitetura plana e com capacidades exorbitantes de escalonamento, o que se encaixa perfeitamente no modelo de repositório ideal para um *data lake*. A compressão dos

Figura 6 – Fluxograma da arquitetura simplificada do data lake da Seazone.



Fonte: Elaborado pelo autor.

dados em Parquet deixa processos de leitura de dados muito mais eficientes, por conta do tamanho reduzido dos arquivos e das propriedades intrínsecas de arquivos Parquet. Dados mais enxutos e legíveis, implicam não só em processos mais rápidos e eficientes, como também mais baratos, por conta dos modelos de pagamento do S3 e do Athena, que seguem a filosofia de *pay-as-you-use* (cobrança flexível envolvendo apenas o que o consumidor efetivamente utilizou).

O *data lake* é constituído de três camadas de armazenamento: *raw*, *clean* e *enriched*, com cada camada sendo representada por um *bucket* diferente.

A camada *raw* retém os dados no seu estado bruto, ou seja, antes de qualquer procedimento de alteração. Apesar de ser possível, as consultas ao *data lake* não costumam ser realizadas nos dados dessa camada, pois ainda não passaram por nenhum processo de preparação para análises. Exceções à regra são: casos onde não é necessário a aplicação de nenhum processo adicional de adequação dos dados, em consultas investigativas de *debug* e em casos de perda ou contaminação de dados em outras camadas.

Após passarem por processos de limpeza e enriquecimento, os dados são armazenados nas camadas *clean* e *enriched*, respectivamente. A grande maioria das análises envolvendo informações presentes no *data lake*, ocorrem através de consultas à esta camada, pois tratam-se de dados tipicamente estruturados de maneira propícia para consultas. Detalhes dos processos de limpeza e enriquecimento não são necessários para o entendimento do PFC, basta a compreensão superficial do propósito de

cada camada do *data lake*.

2.11.2 Aquisição e ingestão de dados

A aquisição dos dados é realizada quase que completamente por *web scrapers* executados em tarefas ECS Fargate. Todos os contêineres desta etapa utilizam uma mesma imagem Docker que é armazenada em um repositório ECR. As únicas diferenças efetivas entre tarefas ECS de diferentes scrapers são: as configurações de tamanho (CPU e RAM) da máquina provisionada, variáveis de ambiente e o comando de CLI executado ao inicializar o contêiner. A imagem Docker em questão. é nada mais que uma réplica do repositório privado no GitHub da Seazone, o "Pipe-scrapers", que abriga o código fonte dos scrapers e *scripts* de auxílio.

Os *scrapers* geralmente são acoplados diretamente a um Kinesis Data Firehose específico, que aponta para o diretório da camada *raw* responsável por abrigar os dados do scraper em questão. Além da inserção no *bucket*, os Firehoses também fazem a conversão e particionamento dos dados adquiridos durante a scrapagem (tipicamente enviados no formato JSON) para Parquet. Existem exemplos de scrapers que realizam a inserção dos dados diretamente pelo endereço do diretório no *bucket*, porém esses casos são raros e apenas aplicáveis em scrapers de baixa escala.

O agendamento e paralelismo dos *scrapers* são realizados através de orquestradores, que são softwares de gerenciamento de fluxos de processos. São os orquestradores que ditam quando, como e com que frequência os *scrapers* são executados. Atualmente, a maioria dos *scrapers* é orquestrada pelo MWAA, porém parte deles já está sendo migrada para o Step Functions.

2.11.3 Entrega de dados

A entrega de dados normalmente ocorre por meio de consultas SQL no Amazon Athena. Para que essas consultas sejam possíveis, o Athena utiliza os metadados do catalogo de dados Glue para obter informações relacionadas ao endereço e esquematização dos dados. Com isso, a *engine* do Athena consegue navegar de maneira eficiente pelo *data lake* e entregar as informações solicitadas. Os metadados das tabelas são atualizados periodicamente por *crawlers* agendados.

2.11.4 Contas AWS da Seazone

O setor de tecnologia da Seazone possui diversas aplicações abrigadas na plataforma AWS, o que exige uma infraestrutura em nuvem abrangente. Por motivos de organização e segurança, a Seazone opta por distribuir suas operações entre várias contas na AWS. Esse é o caso para o *data lake*, onde a parte de aquisição dos dados

se encontra na conta Seazone Technology, enquanto a conta PRD-lake é destinada ao armazenamento, processamento e distribuição desses dados.

3 DESCRIÇÃO DOS PROBLEMAS E REQUISITOS TÉCNICOS

Neste capítulo são detalhados os problemas enfrentados e os requisitos técnicos necessários para a solução proposta. Na Seção 3.1, é discutida a falta de dados externos do Booking.com e as implicações disso. A Seção 3.2 aborda a carência de dados internos da Seazone na plataforma Booking. A Seção 3.3 trata das dificuldades de acesso automático à conta do Booking, enquanto a Seção 3.4 explora os desafios relacionados à detecção de robôs. Finalmente, na Seção 3.5, são apresentados os requisitos técnicos, tanto funcionais quanto não funcionais, que a solução deve atender para resolver os problemas identificados.

3.1 FALTA DE DADOS EXTERNOS DO BOOKING

Conforme descrito na Seção 1.2, a integração bem sucedida da Seazone com o Booking trouxe consigo desafios significativos, causados pela falta de um sistema robusto de aquisição de dados focado nesta plataforma. A ausência de dados externos (dados de listagens fora da posse da Seazone) do Booking limita a capacidade da empresa de ajustar suas estratégias de precificação e de identificar oportunidades de mercado com precisão.

A falta de informações detalhadas sobre preços, disponibilidades e características de imóveis concorrentes, dificulta a realização de análises comparativas eficazes por parte da Seazone, que representariam uma uma forte vantagem competitiva em relação a outros anunciantes.

A perda de valor causada pela falta de coleta destas informações fica ainda mais evidente ao levar em conta o impacto que o Booking tem no mercado de aluguel de hotéis. De acordo com o relatório (PHOCUSWRIGHT, 2022), 52% do valor global bruto arrecadado em reservas de hotéis em 2021 foi obtido por meio de OTAs, sendo que, segundo (MARTIN-FUENTES; MELLINAS, 2018), o Booking representa cerca de dois terços do mercado de OTAs para hotéis, solidificando a importância da plataforma no ramo de aluguel por estadia de hotéis, estatísticas significativas considerando que quartos de hotéis/resorts como Il Campanário e Jurerê Beach Village, fazem parte fundamental do repertório de imóveis da Seazone.

3.2 FALTA DE DADOS INTERNOS DO BOOKING

Por motivos de desempenho e manejamento, a Seazone aplica uma série de processos de garantia de qualidade e padronização em seus anúncios, o que torna essencial a monitoração continua de listagens nos sites de OTAs para garantir que os processos estão sendo seguidos. Isso é feito através da verificação de informações como regras da casa, avaliações e comodidades nas páginas de aluguel de estadia.

Dado o grande número de imóveis sob o gerenciamento da Seazone, não é eficiente realizar esse monitoramento de forma manual. Portanto, a obtenção automatizada dos dados das próprias listagens da Seazone é uma solução interessante, pois permite a automação dos processos de verificação, assegurando a conformidade contínua e eficiente com os padrões de qualidade estabelecidos.

Uma solução ideal para o problema da falta de dados internos de listagens da Seazone no Booking, seria a obtenção de dados a partir de APIs fornecidas pela própria plataforma, o que não é uma prática incomum indústria. Contudo, atualmente a Seazone não tem acesso a tal API, o que torna necessário a procura de métodos diferentes para obter esses dados de forma eficiente e precisa, com o desenvolvimento de web *scrapers* autorais sendo um desses métodos.

3.3 DIFICULDADE DE ACESSO AUTOMÁTICO À CONTA DO BOOKING

Assumindo que o desenvolvimento de web *scrapers* capazes de adquirir os dados de interesse da plataforma foi um sucesso, ainda seria necessário um processo para obter periodicamente informações essenciais dos imóveis da Seazone, como URLs e IDs do Booking, para que os scrapers saibam quais anúncios deverão ser processados. No caso do Airbnb, os *scrapers* acessam (de maneira confidencial) automaticamente a conta da Seazone e atualizam diariamente a lista contendo as informações necessárias para a *scrapagem* de imóveis ativos. No entanto, isso não é viável para o site do Booking, pois é necessário passar por uma autenticação em duas etapas para acessar a parte logada da Seazone, o que implicaria em um comprometimento de segurança ou na adição de uma etapa manual, caso o acesso automático fosse tentado, dificultando a automação completa do processo.

3.4 DETECÇÃO DE ROBÔS

Tal como detalhado na Seção 2.5, o Booking, como uma grande plataforma web, utiliza sistemas de detecção de robôs implementados para proteger a integridade de seu site e garantir que o seu tráfego seja ao máximo possível proveniente de usuários reais. Mecanismos de limitação de velocidade de acesso, verificações de elementos HTTP e bloqueio de IPs por lista negra, podem retardar ou até impedir o funcionamento de web *scrapers*, dificultando a coleta automatizada de dados, especialmente nos volumes necessários para análises detalhadas.

3.5 REQUISITOS TÉCNICOS

Requisitos funcionais especificam as funcionalidades e comportamentos que um sistema deve ter para atender às necessidades dos usuários. Eles descrevem o

que o sistema deve fazer, incluindo tarefas e processos necessários.

Com base na descrição dos problemas abordados ao longo deste capítulo, combinado com a experiência do time de dados da Seazone com sistemas de aquisição e processamento de dados provenientes de web *scrapers* em outras OTAs, chegou-se a conclusão de que o projeto deverá ser um sistema de aquisição de dados por web *scraping* que seja capaz de cumprir os requisitos funcionais listados a seguir:

- **Extração de dados importantes:** O sistema deve ser capaz de extrair os dados de maior relevância para processos envolvendo anúncios do Booking, desde que esses dados sejam capturáveis pela página de reserva de cada anúncio;
- **Armazenamento dos dados:** Os dados extraídos pelo sistema devem ser armazenados no *data lake* da Seazone para que possam ser integrados ao restante da infraestrutura de dados;
- **Acessibilidade dos dados:** Os dados armazenados devem ser facilmente consultáveis por processos e usuários que tenham acesso ao *data lake*;
- **Abrangência:** O sistema deve adquirir os dados de todas a listagens ativas da Seazone na plataforma Booking;
- **Automação:** A aquisição de dados deve ser completamente automática, sem a necessidade de intervenção humana em nenhuma das etapas;

Requisitos não funcionais descrevem como o sistema deve se comportar, se concentrando nos atributos de qualidade e nas restrições de funcionamento do sistema, sem especificar funcionalidades específicas. A partir disso, foram levantados os seguintes requisitos não funcionais para o projeto:

- **Desempenho e escalonabilidade:** O sistema deve ser capaz de aquirir dados de uma grande quantidade de anúncios em tempo hábil, ou seja, rápido o suficiente para que possa ser feito a expansão futura do número de listagens *scrapadas* sem demorar dias para realizar a aquisição completa;
- **Custo:** O sistema deve ser desenvolvido levando em conta a otimização de custos, aproveitando os recursos de forma eficiente;
- **Centralização de códigos:** Os códigos utilizados no projeto devem ser centralizados e organizados em um único local do qual a equipe de dados tenha acesso, facilitando a manutenção e a implementação novas funcionalidades;

4 EXPLORAÇÃO DAS PLATAFORMAS E SOLUÇÃO PROPOSTA

Este capítulo explora as plataformas utilizadas e apresenta a solução proposta para os desafios identificados. Na Seção 4.1, é descrito como será feito a extração de dados das plataformas Booking e Stays. A Seção 4.2 apresenta a solução proposta, detalhando a arquitetura do sistema de aquisição e processamento de dados.

4.1 EXPLORAÇÃO DAS PLATAFORMAS

Antes de arquitetar o sistema de web *scraping*, é necessário o levantamento de quais dados possuem a maior importância para as análises mais comuns da Seazone e quais deles são efetivamente adquiríveis pelo site do Booking.com. Baseado em discussões com o time de precificação (é o time onde os dados são mais impactantes) e em processos de monitoramento de qualidade realizados pelo time de operação, foi obtido a lista de informações importantes presentes no Booking apresentada a seguir:

- Título do anúncio e dos imóveis;
- Localização do anúncio;
- Avaliações do anúncio;
- Comentários do anúncio;
- Comodidades dos imóveis;
- Regras da casa do anúncio;
- Contagem de cômodos dos imóveis;
- Dados de preço e disponibilidade dos imóveis.

Ao contrário do Airbnb, o Booking permite o agrupamento de múltiplos imóveis (ou quartos, no caso de hoteis) em único anúncio, esse é o motivo da distinção entre dados referentes à anúncios e imóveis na lista recém apresentada.

4.1.1 Metodologia de exploração

Web *scrapers* utilizam informações públicas disponibilizadas em sites para extrair dados relevantes. Para determinar quais informações são capturáveis, é essencial realizar uma exploração detalhada das páginas alvo. Esse processo é frequentemente auxiliado pelo uso do Developer Tools (Ferramentas de Desenvolvedor) disponível nos navegadores modernos.

O Developer Tools é um conjunto de ferramentas embutidas nos navegadores que permitem os desenvolvedores inspecionarem e modificarem páginas da web. Ele

é acessível via o menu do navegador ou com a tecla F12 e possui uma variedade de funcionalidades úteis para o usuário. No contexto do atual PFC, a exploração dos sites foi feita no navegador Google Chrome com o auxilio das seguintes funcionalidades do Developer Tools:

- **Inspecionar elemento:** Permite identificar a estrutura HTML da página e localizar exatamente onde os dados estão situados, facilitando a identificação de quais dados são adquiríveis pelo HTML estático;
- **Interações de rede:** Monitoramento de todas as requisições feitas pela página, revelando *endpoints* de APIs e outras interações úteis para extração de dados. Usado na descoberta de quais dados são retornados por APIs e de que forma as chamadas são realizadas pelo navegador;
- **Cache de aplicações:** Acesso aos dados armazenados localmente, como *cookies* e *tokens* de armazenamento local, que podem ser necessários para autenticação e identificação de padrões de fluxo de informação.

4.1.2 Avaliações do anúncio

Os dados de avaliação dos anúncios são obtidos diretamente da página de reservas, através da aba de avaliações. A aba pode ser observada na parte direita da captura de tela da Figura 7, e ela é aberta com o pressionar do botão “ler todas as avaliações” (circulado em amarelo) ou adicionando “#tab-reviews” no final da URL da página (sublinhado em vermelho).

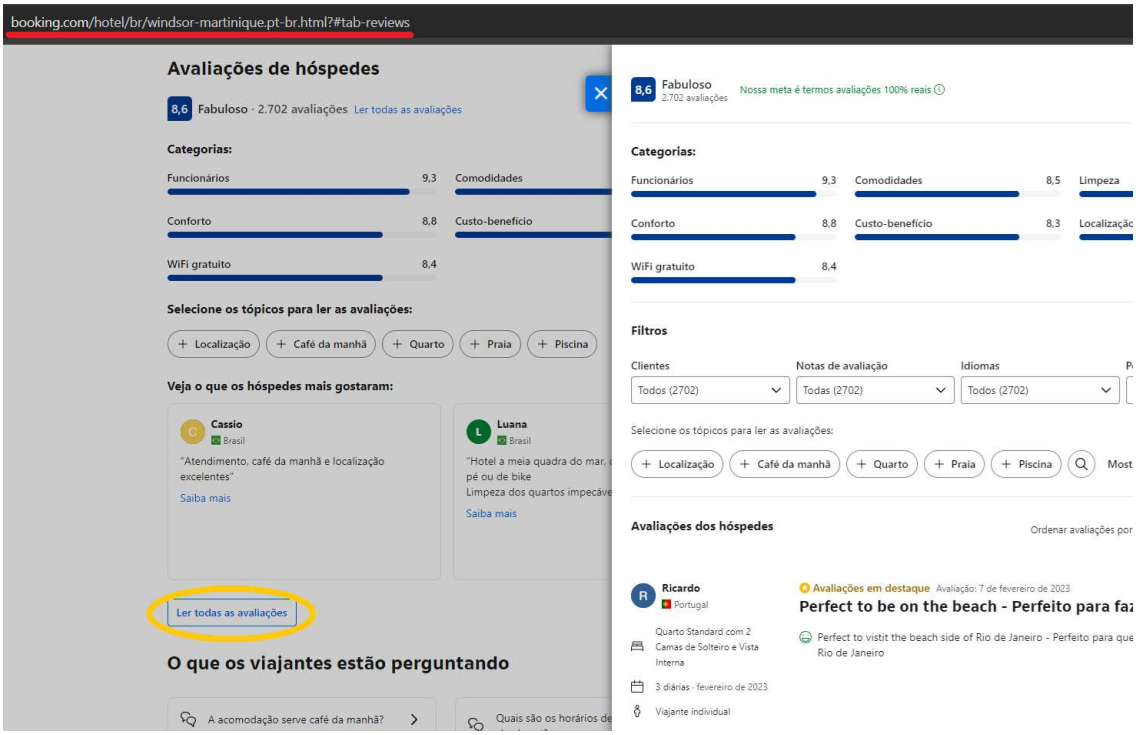
O procedimento de pressionamento do botão pode ser replicado por código através de técnicas de emulação de navegadores, com uma maneira popular sendo o uso da biblioteca Selenium do Python. Também é possível obter os dados por uma requisição HTTP ao URL alterado da página. Nos dois casos, as informações de avaliação estão presentes no HTML estático retornado, portanto, podem ser adquiridas por *scrapers* desde que eles tenham acesso ao URL do anúncio.

4.1.3 Comentários do anúncio

Os dados de comentários também estão localizados na aba de avaliações, conforme mostrado na Figura 8. Contudo, apesar dos comentários estarem presentes no HTML desta aba, a paginação não é possível de ser feita com uma simples manipulação de URL.

Explorando a aba um pouco mais a fundo, foi descoberto que o clique nos botões de paginação (sublinhados em vermelho na Figura 8) faz uma requisição a uma API que retorna o HTML dos comentários da página solicitada, sendo apenas necessário o URL original do anúncio para que a API funcione. Assim, os dados de comentários do

Figura 7 – Captura de tela da aba de avaliações do Booking.



Fonte: Autor.

anúncio podem ser obtidos tanto por emulação de navegadores, quanto por chamas APIs.

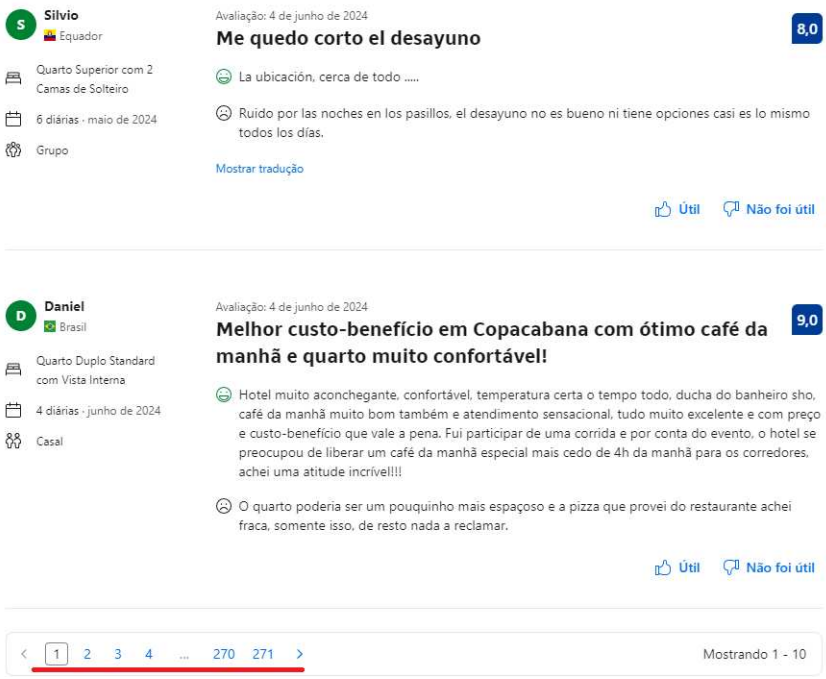
4.1.4 Regras da casa do anúncio

Os dados de regras da casa são apresentados na aba principal da página do anúncio, conforme exposto na Figura 9. Por conta disso, essas informações são facilmente adquiridas através da navegação do HTML estático, que é retornado por uma requisição HTTP ao URL da página principal do anúncio.

4.1.5 Detalhes dos imóveis

A Figura 10 mostra a aba contendo o título e as comodidades de um imóvel específico, que é acessada por um clique em algum dos links circulados em vermelho. Essas informações podem ser adquiridas através de emulação de navegador, seguido de uma extração de dados em HTML. Posto isso, essas informações também são passadas em formato JSON pela API que é chamada com o clique dos links. Adicionalmente, a resposta da API também contém o título e cidade do anúncio, além de informações descritivas dos cômodos do imóvel selecionado, tornando o uso da API uma alternativa bem mais atraente.

Figura 8 – Captura de tela dos comentários na aba de avaliações do Booking.



Fonte: Autor.

Para a aquisição dos dados por API, é preciso passar o ID do anúncio e o ID individual do imóvel pela requisição. Esses IDs são identificadores únicos de uso interno do Booking e tipicamente não são expostos aos usuários normais.

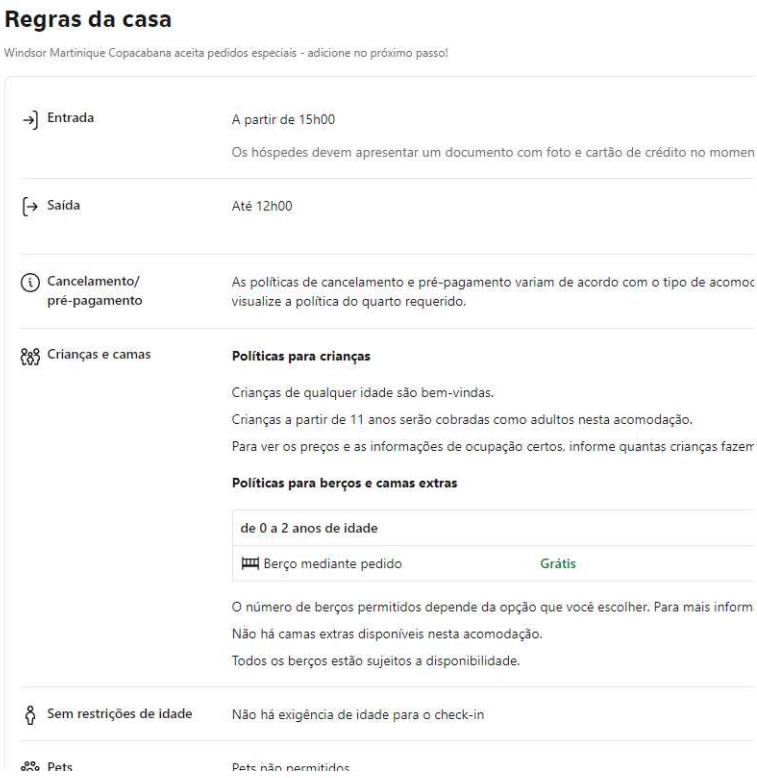
4.1.6 Preço e disponibilidade dos imóveis

Embora sejam de grande importância para a empresa, não foi possível achar métodos adequados de extração automática de preço e disponibilidade.

O formato de dados de preço usado pela Seazone, consiste no custo de uma estadia de um único dia, de cada dia, de cada imóvel, para pelo menos alguns meses no futuro. Para obter esse tipo de dado, é necessário ter algum campo de preço com as propriedades: (i) contém o preço de exatamente uma diária; (ii) esse preço aparece consistentemente em um número razoável de datas futuras; (iii) é o preço específico de um imóvel em particular; (iv) o campo existe para todos os imóveis;(v) sabe-se de qual imóvel é o preço.

A primeira vista, o calendário de reservas (exposto na Figura 11) parece cumprir esses requisitos, porém é apenas mostrado o preço da menor diária de cada dia, o que só funcionaria para anúncios de um único imóvel. Também foi experimentado a tentativa de extrair o preço através da seleção em sequência de estadias de um dia só, para extrair os dados da seção de visualização de preços (botões azuis da Figura 11),

Figura 9 – Captura de tela das regras da casa de uma listagem do Booking.



Fonte: Autor.

porém apenas imóveis com estadia mínima de um dia aparecem neste método.

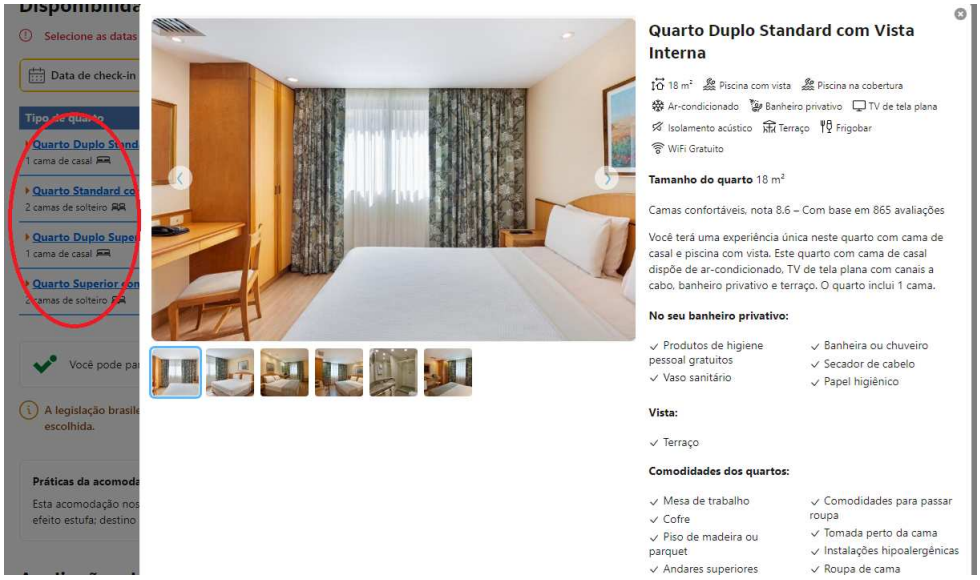
Foram tentados métodos similares para a extração de disponibilidade, porém uma data só é dada como indisponível no calendário, caso não tenha nenhum imóvel disponível no dia, invalidando esta possibilidade. Métodos envolvendo a seleção em sequência de estadias também não funcionam de maneira eficiente, pois só é possível saber se um imóvel realmente está indisponível, caso tenha-se acesso à informação de estadia mínima de antemão (que é um dado individual de cada imóvel), pois o imóvel pode não aparecer como uma opção alugável pelo simples fato dele não aceitar estadias com o tamanho selecionado.

4.1.7 URLs e IDs de anúncios da Seazone

Para que todos os *scrapers* sugeridos possam funcionar, as URLs e os IDs Booking de anúncios Seazone precisam ser coletados automaticamente de algum lugar. Conforme explicado na Seção 3.3, isso seria tradicionalmente feito acessando a conta da Seazone no Booking e *scrapando* os dados dali, porém o login com autenticação em duas etapas do Booking não permite isso. A solução proposta para este problema é o uso da Stays como ponto de coleta das informações necessárias.

Tal como detalhado na Seção 2.2, a Seazone utiliza a Stays como ponto de

Figura 10 – Captura de tela da aba de detalhes de imóvel no Booking.



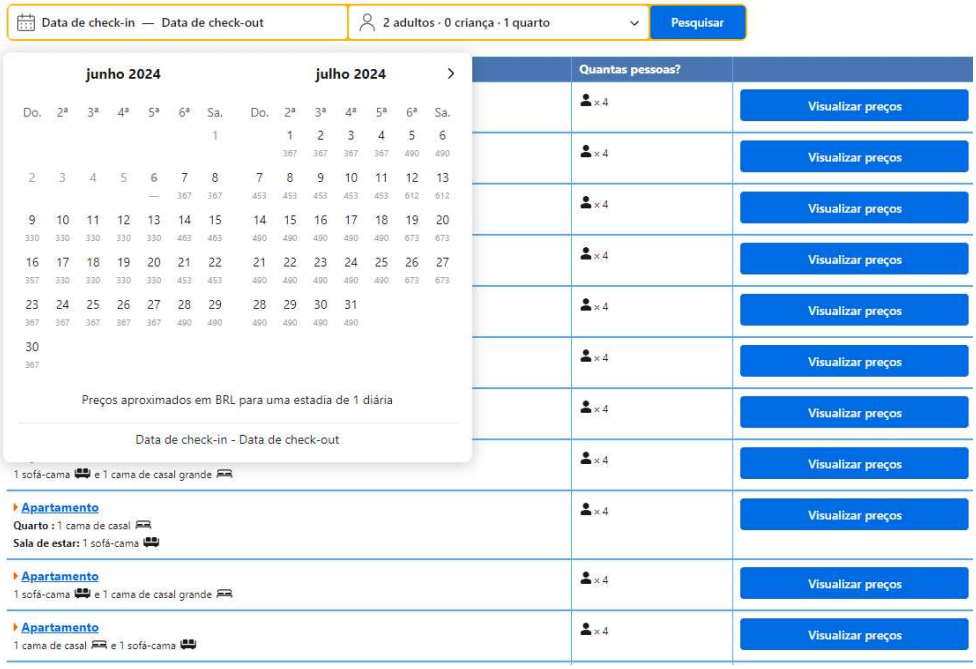
Fonte: Autor.

sincronia entre seus imóveis em diferentes OTAs. Por conta disso, todas as listagens do Booking estão conectadas diretamente com a plataforma. Após navegar na Stays aplicando a mesma metodologia de exploração utilizada no restante dos *scrapers*, foi achado a página exposta na Figura 12. Nela encontram-se todos os imóveis ativos e inativos que a Seazone tem no Booking. Ao pressionar o botão circulado em vermelho na Figura 12, é aberto a aba específica do anúncio mostrado, e circulado em amarelo, está um link direto à página do anúncio no Booking.

É possível replicar a funcionalidade do botão circulado em vermelho na Figura 12 com apenas dados do HTML da própria página de anúncios, executando a mesma requisição HTTP de API que ele chama e retornando as informações da aba do imóvel, também em HTML. Dentre os dados retornados, está a mesma URL de anúncio que o botão circulado em amarelo utiliza para levar o usuário até o site, além do ID interno do Booking referente à listagem em questão. Com isso, basta implementar um *scraper* que automatize o processo de coleta de IDS e URLs a partir dos HTMLs citados, que o problema de falta de *inputs* para o restante *scrapers* é resolvido.

O que torna viável a implementação deste *scraper* da Stays, mesmo com ele tendo que entrar em uma área logada para funcionar, é o fato da Stays providenciar mecanismos de criação de usuários com permissões limitadas o suficiente. Assim, o *scraper* pode utilizar uma conta feita especificamente para ele, que apenas tem acesso de leitura das páginas que ele precisa acessar, o que abre consideravelmente menos brechas de segurança do que a alternativa análoga na conta do Booking.

Figura 11 – Captura de tela do calendário de reservas do Booking.



Fonte: Autor.

4.2 SOLUÇÃO PROPOSTA

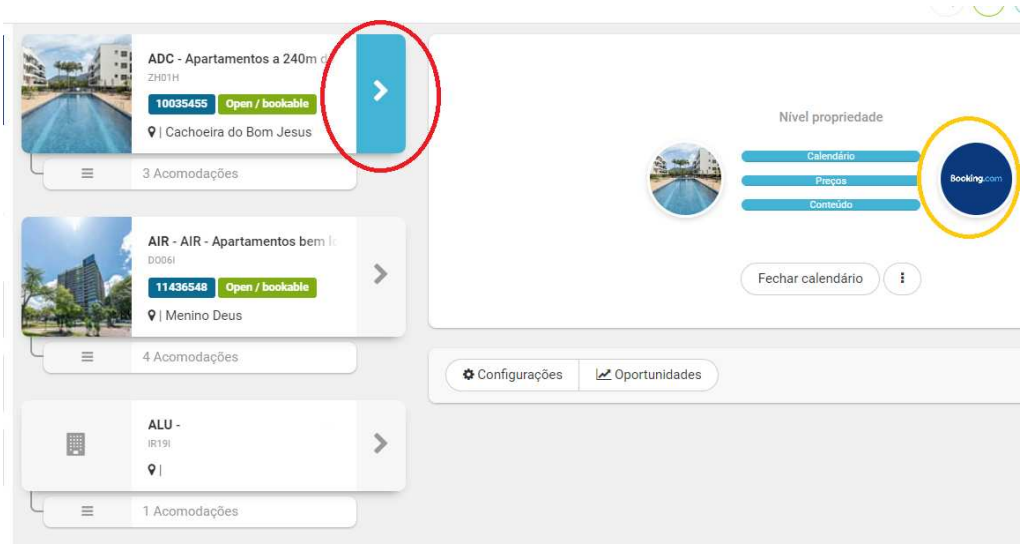
Com base nos problemas e requisitos técnicos abordados no Capítulo 3, e na exploração de sites descrita na Seção 4.1, foi projetado o sistema de aquisição, armazenamento e disponibilidade de dados do Booking, representado pelo fluxograma da Figura 13.

Para seguir a mesma lógica organizacional do restante do *data lake*, o sistema será separado em duas contas AWS. As etapas relacionadas à aquisição de dados ficarão na conta Seazone Technology, enquanto as de processamento, armazenamento e disponibilização de dados serão alocadas na conta PRD-Lake.

O sistema começa com o *scraper* da Stays, que vai coletar os URLs e IDs de anúncios ativos da Seazone no Booking. Os dados coletados serão convertidos em arquivos JSON e depositados diretamente em um *bucket* S3 utilizado para o armazenamento de *inputs* de *scrapers*. Esse *bucket* também estará localizado na conta Seazone Technology para que os *scrapers* possam ter fácil acesso a ele. Apesar de não ser necessário por conta da sua simplicidade, a execução do *scraper* da Stays será agendada através do Step Functions, dessa maneira ele ficará no mesmo padrão dos demais *scrapers* do sistema.

Os *scrapers* do Booking (cuja arquitetura será descrita com mais detalhes na Subseção 4.2.1) vão adquirir as informações de interesse do Booking.com a partir

Figura 12 – Captura de tela da página da Stays com conexão aos anúncios da Seazone no Booking.



Fonte: Autor.

dos *inputs* resultantes de *scraping* na etapa anterior. Cada *scraper* do Booking será atrelado a um Firehose exclusivo, ao qual mandará os dados coletados formatados em JSON.

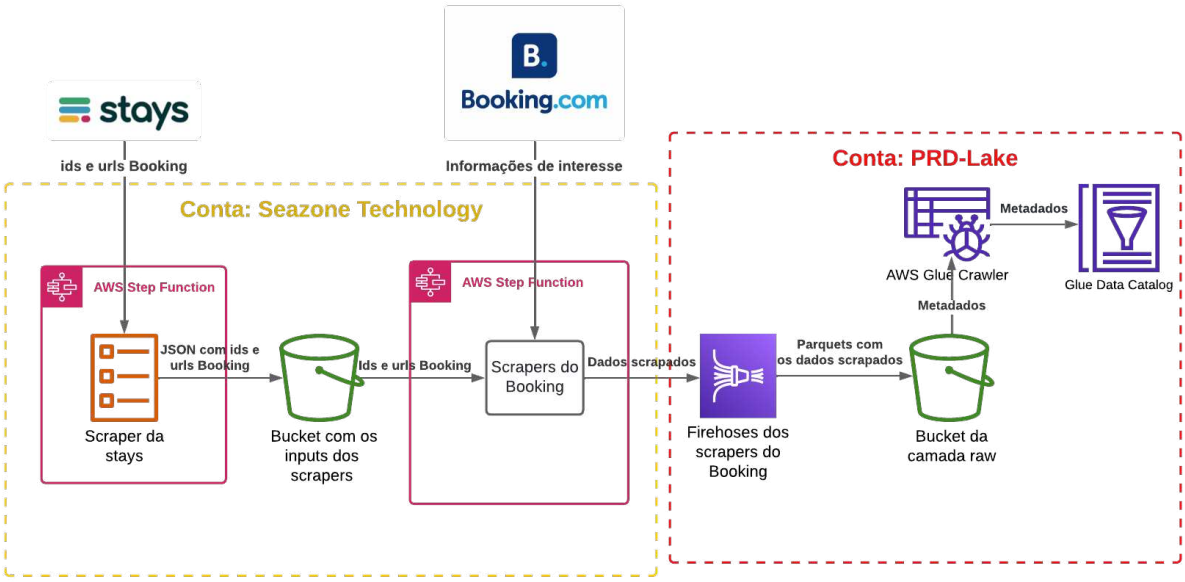
Cada Firehose será conectado a um diretório da camada *raw* correspondente a uma tabela Glue. Os Firehoses exercerão a função de compactar os dados recebidos em arquivos Parquet, que serão armazenados e particionados em seus respectivos diretórios.

Cada tabela Glue terá um *crawler* específico responsável por atualizar os metadados necessários para sua consulta. Os *crawlers* serão configurados para que executem logo após o termino da *scrapagem* da sua tabela, permitindo o acesso constante dos dados coletados via Athena. Sem esta etapa, toda vez que uma partição nova fosse adicionada em qualquer um dos diretórios de dados, a sua respectiva tabela Glue não seria capaz de consultá-la, pois a existência desta nova partição não estaria em seus metadados.

4.2.1 Arquitetura de scrapers do Booking

A extração de dados do Booking é dividida em quatro categorias distintas de *scrapers* com base nas informações que coletam: detalhes, comentários, avaliações e regras da casa. Essa separação é feita por alguns motivos. Primeiramente, o desacoplamento dos *scrapers* faz com que a quebra de um não afete o funcionamento de outro, tornando o sistema consideravelmente mais robusto, com o mesmo podendo ser dito para os serviços AWS anexados à cada *scraper*. Ela também permite que cada

Figura 13 – Fluxograma da arquitetura em nuvem da solução proposta.



Fonte: Autor.

scraper tenha as configurações mais adequadas para seu funcionamento, visto que as fontes de dados são diferentes, cada método de extração de informações pode acabar exigindo capacidades computacionais e peculiaridades diferentes. Por fim, essa separação também permite a expansão do sistema com mais fluides, facilitando a adição de futuros novos *scrapers* do Booking sem ter que alterar os antigos.

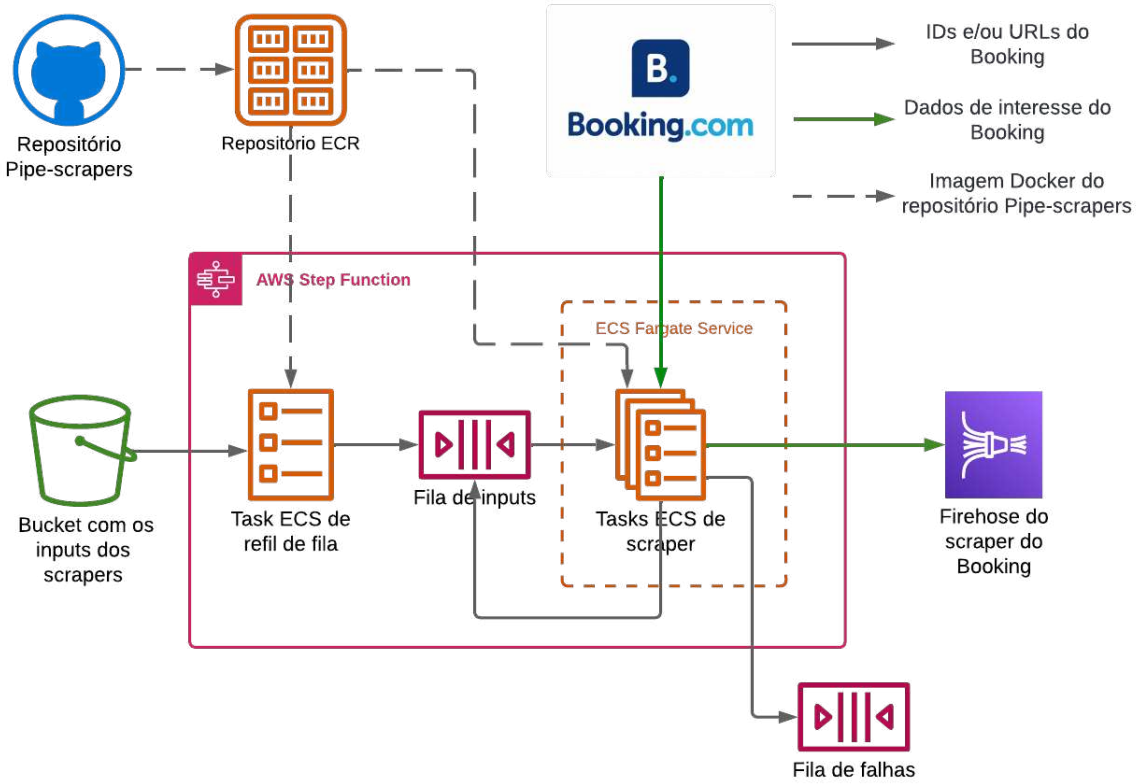
A arquitetura proposta para a extração de dados do Booking é exemplificada na Figura 14 e representa o funcionamento generalizado de cada *scraper* do Booking.

O sistema começa com a execução de um *script* de refil de filas. Ele vai fazer o download do arquivo JSON com as entradas do *scraper* e os insere na fila de *inputs*, com cada mensagem da fila representando os *inputs* necessários para a aquisição dos dados de um único anúncio. Esse *script* é executado através de uma única tarefa ECS.

Após o *script* de refil terminar de preencher a fila, é iniciado a etapa de web *scraping*. O serviço ECS responsável por gerenciar as tarefas dos *scrapers* é acionado para manter um número específico deles em execução simultânea, com todos estando conectados às mesmas filas SQS de entrada e falha de mensagens. O processo é executado seguindo o método SIMD, previamente detalhado na Seção 2.9, sendo que aqui a unidade de controle é a máquina de estado configurada no Step Functions, que manterá o comando de execução de *scrapers* até que a fila de *inputs* seja esvaziada.

A interação dos *scrapers* com as filas SQS é ditada pela biblioteca Sqs.py, que

Figura 14 – Fluxograma da arquitetura proposta para a extração de dados do Booking.



Fonte: Autor.

estabelece um sistema de consumo e retentativa de mensagens. Os *inputs* da fila SQS são constantemente ingeridos pelos *scrapers* que, por sua vez, tentam extrair os dados do Booking dos anúncio referenciados pelos *inputs*. Caso a listagem recebida seja *scrapada* com sucesso, os dados coletados são mandados para o Firehose e a mensagem é apagada. Nas situações onde o *scraper* não consegue extrair os dados, é incrementado um contador de tentativas inerente à própria mensagem, e é tomado uma decisão: se o número de tentativas tiver passado do valor máximo estipulado, a mensagem é considerada defeituosa e mandada para fila de falhas, caso contrário, ela é apenas retornada à fila de *inputs* com um novo valor em seu contador.

Quando uma mensagem não é processada corretamente por um *scraper*, significa que algum problema ocorreu com a máquina executando tal *scraper* ou com a listagem que ele está consumindo. Identificar a causa específica pode ser desafiador, pois duas situações comumente experienciadas são: o bloqueio de máquinas AWS e a ingestão de listagens inativas. O bloqueio de máquinas ocorre com alta frequência, devido aos sistemas de detecção de robôs que captam facilmente requisições vindas

de IPs da AWS, momento em que normalmente são bloqueadas do site, fazendo com que o *scraper* não consiga mais coletar os dados de nenhum anúncio até que seja reiniciado. Nos casos onde o *scraper* consome uma listagem inativa, ele também não consegue coletar os seus dados, porém dessa vez o problema está na própria mensagem, o que invalida a solução de reinicializar o *scraper*. O sistema de retentativas combate justamente esse tipo de situações, permitindo a identificação de onde está o problema, mesmo em casos onde essa resposta possa parecer ambígua.

Ambas as tarefas ECS executadas nesta etapa são executadas a partir de uma única imagem Docker contendo todos os *scrapers* em atividade da Seazone (incluindo o *scraper* da Stays). Esta imagem é uma réplica do repositório Pipe-scrapers do GitHub da Seazone, que atualiza o repositório ECR toda vez que uma nova versão do repositório é lançada. Este processo é feito através do GitHub Actions, que é um recurso do GitHub para a automação de fluxos de trabalho. Dessa forma, os *scripts* de web *scraping* sempre estão sincronizados uns com os outros, além de facilitar o desenvolvimento contínuo deles.

5 DESENVOLVIMENTO

Este capítulo detalha o desenvolvimento do sistema de aquisição e processamento de dados do Booking.com. A metodologia de desenvolvimento dos *scrapers* é apresentada na Seção 5.1. A Seção Seção 5.2 aborda a seleção dos métodos de web *scraping*. Na Seção 5.3, é descrita a implementação local dos protótipos dos *scrapers*, seguida pela Seção 5.4, que trata do desenvolvimento do *script* de refil de inputs. A integração dos *scrapers* locais com a nuvem é explicada na Seção 5.5. A Seção 5.6 descreve a implementação e teste dos *scrapers* em ambiente EC2. A implantação do sistema em ECS e sua automação completa são discutidas em Seção 5.7 e Seção 5.8, respectivamente. A Seção 5.9 detalha a implementação do *scraper* de IDs e URLs Booking pela Stays.

5.1 METODOLOGIA DE DESENVOLVIMENTO DOS SCRAPERS DO BOOKING

Foi adotado uma metodologia sistemática de desenvolvimento para os *scrapers* do Booking, pois, apesar das diferenças específicas entre eles, todos compartilham uma base comum de funcionamento e utilizam ferramentas similares. Por isso, a criação de uma metodologia padronizada permitiu organizar o processo de maneira eficiente. Isso assegura que cada *scraper* seja desenvolvido de forma consistente, reduzindo a complexidade e facilitando a manutenção e expansão futura do sistema.

A metodologia de desenvolvimento dos *scrapers* do Booking, da concepção até a implementação em ambiente de produção, é detalhada nas etapas a seguir:

1. **Exploração dos sites à procura de informações coletáveis:** Etapa com o objetivo de descobrir quais dados são coletáveis através da plataforma Booking, e de que maneiras a extração deles pode ser automatizada. Esta etapa já foi documentada na Seção 4.1, pois era relevante para o desenvolvimento e justificativa da solução proposta;

2. **Seleção do método de web *scraping*:** Com base nos aspectos positivos e negativos dos métodos levantados, é escolhido de que maneira cada *scraper* fará a aquisição dos dados;

3. **Implementação local do protótipo do *scraper*:** Utilizando o método selecionado, é implementado localmente uma versão básica do código do *scraper*. Essa etapa serve para montar a parte lógica de extração de dados, sem se preocupar muito com o paralelismo de processos e interações com a nuvem;

4. **Implementação do *script* de refil de *inputs*:** O *script* de refil é implementado diretamente no *framework* de execução do repositório Pipe-scrapers e as filas SQS de *inputs* são criadas na AWS;

5. **Integração do *scraper* local com a nuvem:** O código do *scraper* é convertido para o *framework* de execução do Pipe-scrapers, é feito a criação e conexão do restante dos recursos de integração em nuvem (fila SQS de falhas, diretórios no S3, Firehose, *crawler* e tabela Glue) e é atualizado o repositório remoto do GithHub com as alterações necessárias para o funcionamento do *scraper*;

6. **Teste do processo em EC2:** Nesta etapa, todos os passos do sistema de web *scraping* são testados em uma instância EC2 que simula a execução em nuvem do *scraper*. Esse procedimento é realizado para achar e corrigir possíveis problemas envolvendo mecanismos anti-robô com mais agilidade, pois costumam ser mais rigorosos com tráfego proveniente da nuvem;

7. **Implementação do sistema de web *scraping* em ECS:** Após validar o funcionamento do *scraper* em nuvem, a imagem Docker com o código dos *scrapers* é atualizada no ECR, e, em seguida, é feito a criação das tarefas e serviços ECS necessárias para a execução do sistema;

8. **Automação completa do sistema em produção:** Por fim, cria-se a máquina de estados responsável pelo agendamento e orquestração em produção do sistema completo de aquisição de dados referente ao web *scraper* recém desenvolvido.

Nas seções a seguir, será detalhado o desenvolvimento de cada etapa da metodologia.

5.2 SELEÇÃO DOS MÉTODOS DE WEB *SCRAPING*

A seleção dos métodos de web *scraping* considerou a eficiência e a praticidade das diferentes abordagens.

As combinações de métodos viáveis de busca e formato de dados retornados, que foram levantados na Seção 4.1, podem ser categorizados de acordo com a Tabela 1.

Alternativa	Método de Busca	Formato de dados
1	Emulação de navegador por código	HTML
2	Chamada HTTP usando a URL da página	HTML
3	Chamada HTTP ao *endpoint* de uma API	HTML
4	Chamada HTTP ao *endpoint* de uma API	JSON

Tabela 1 – Tabela de alternativas de combinações de métodos viáveis de busca e formato dos dados retornados.

Dados em JSON possuem uma estrutura de chave-valor simples e direta, permitindo uma extração extremamente eficiente de dados por parte do computador, mesmo em arquivos de grande porte. Em contrapartida, o HTML, requer interpretadores mais

complexos para navegar em sua estrutura hierárquica, o que aumenta o tempo de processamento e a carga computacional. Por conta desses fatores, alternativas envolvendo JSON foram priorizadas.

A emulação de navegador é consideravelmente mais custosa em termos de recursos computacionais e tempo de execução, sendo mais adequada apenas em cenários com sistemas anti-robô rigorosos, onde sua capacidade de simular o comportamento humano pode evitar bloqueios e detecções indesejadas. Portanto, alternativas envolvendo este método foram inicialmente desconsideradas.

Por meio dessas análises, chegou-se na seguinte cadeia de prioridades entre alternativas: 4 > 2 ou 3 > 1. Posto isso, nenhum grupo de informações a serem *scrapadas* possui ambas as alternativas 2 e 3 como possibilidade. Portanto, baseado na cadeia de prioridades postulada e nos métodos disponíveis para a extração de cada tipo de dado, tem-se a seleção dos métodos de web *scraping* a seguir:

1. ***Scraper* de detalhes:** Busca de dados por chamada HTTP ao *endpoint* da API e extração de dados formatados em JSON;
2. ***Scraper* de comentários:** Busca de dados por chamada HTTP ao *endpoint* da API e extração de dados formatados em HTML;
3. ***Scraper* de avaliações e *scraper* de regras da casa:** Busca de dados por chamada HTTP usando a URL de uma página pública e extração de dados formatados em HTML.

5.3 IMPLEMENTAÇÃO LOCAL DOS PROTÓTIPOS DOS *SCRAPERS*

Nesta etapa é feito um *script* em Python rudimentar, capaz de extrair e armazenar localmente os dados do Booking a partir de uma lista compreensível de *inputs*. Um exemplo representativo da lógica simplificada destes códigos pode ser visto na Figura 15.

O *script* basicamente itera pelos *inputs* da lista local e em cada interação ele: executa a chamada à página ou API alvo utilizando a biblioteca *requests*, extrai as informações de interesse dos dados retornados pela chamada e os adiciona em formato de dicionário a uma lista de dados coletados. Por fim, ele armazena o que foi coletado em um JSON para análise posteriores de integridade dos dados.

O JSON de *inputs* dos *scrapers* consiste de uma lista de dicionários contendo uma chave para URL e outra para o ID Booking do anúncio, conforme exemplificado na Figura 16. Os testes locais desta etapa foram feitos utilizando um arquivo de *inputs* armazenado na mesma pasta do protótipo, devido ao fato do *script* ainda não ter sido integrado com a nuvem.

Figura 15 – Código representativo da lógica simplificada de um *scraper* do Booking.

```python
import json

def requisicao_do_anuncio(anuncio):
    # Lógica de requisição dos dados do anúncio
    return # Dados retornados pela requisição

def extracao_dos_dados(requisicao):
    # Lógica de extração dos dados do anúncio
    return # Dados extraídos do anúncio

with open('inputs_de_anuncios.json', 'r') as f:
    inputs = json.load(f)

dados_coletados = []

for anuncio in inputs:
    requisicao = requisicao_do_anuncio(anuncio)
    dados_do_anuncio = extracao_dos_dados(requisicao)
    dados_coletados.append(dados_do_anuncio)

with open('dados_coletados.json', 'w') as f:
    inputs = json.dump(dados_coletados, f)
```

Fonte: Autor.

5.3.1 *Scrapers* com extração de dados em HTML

Os *scrapers* de comentários, avaliações e regras da casa, tem como retorno de suas chamadas de busca, dados no formato HTML. Por conta disso, nos três casos, a extração das informações de interesse é feita com o auxílio da biblioteca BeatifulSoup.

Figura 16 – Exemplo de um JSON de *inputs* contendo três imóveis do Booking.


```json
[
  {
    "id_booking": "10035455",
    "booking_url": "https://booking.com/hotel/br/moderno-apto
      -2min-da-praia-c-47-churrasq-adc105.html"
  },
  {
    "id_booking": "11436548",
    "booking_url": "https://booking.com/hotel/br/air
      -apartamentos-bem-localizados-em-porto-alegre-rs.html"
  },
  {
    "id_booking": "10731993",
    "booking_url": "https://booking.com/hotel/br/are
      -apartamentos-proximos-ao-centro.html"
  }
]
```


Fonte: Autor.

Tal como descrito na Seção 2.10, o BeautifulSoup atua através da navegação de uma árvore de análise instanciada por um interpretador sintático. A Figura 17 exemplifica o uso desta biblioteca para a extração dos dados de avaliações, onde, na linha 97, é feito a inicialização da árvore de análise utilizando o "lxml", o interpretador mais eficiente em termos processamento de HTML que o BeatifulSoup providencia.

Figura 17 – Segmento de código ilustrando o funcionamento do BeatifulSoup na extração de dados.

```
97      soup = BeautifulSoup(req.text, features="lxml")
98
99      ratings_titles = soup.find_all("p", {"class": "review_score_name"})
100
101 >    ratings_dict = {···
109
110     if not ratings_titles: return ratings_dict
111
112     for ratings_title in ratings_titles:
113
114         if ratings_title.text in self._ratings_mapping.keys():
115
116             score_value = ratings_title.parent.find("div", {"class" : "score_bar_value"})['data-score']
117             score_value = int(score_value)/10
118
119             ratings_dict[self._ratings_mapping[ratings_title.text]] = score_value
120
121     ratings = soup.find("div", {"class" : "abf093bdfe f45d8e4c32 d935416c47"}).text
122     regex = re.compile('(.+) ava')
123     ratings_dict['number_of_ratings'] = int((regex.search(ratings).group(1)).replace('.',''))
```

Fonte: Autor.

A extração dos dados é feita primariamente pelas funções "find" e "find_all" da biblioteca, que permitem a localização de elementos HTML específicos a partir de suas características. A partir da exploração da página do HTML, é possível achar padrões de formatação, que podem ser aproveitados pelas funções de busca citadas para obter as informações de interesse de forma replicável. A Figura 17 mostra exatamente isso, onde, no código mostrado, tira-se proveito do padrão organizacional das categorias de avaliação para extrair as notas do anúncio iterativamente.

5.3.2 *Scraper* de detalhes

O *scraper* de detalhes tem algumas diferenças significantes em seu funcionamento comparado aos outros. A primeira é o fato dos dados desse *scraper* virem no formato JSON. Isso torna o processo de extração muito mais fácil e eficiente, pois os dados brutos retornados vão constituir de apenas listas e dicionários, estruturas facilmente navegáveis em Python.

Uma característica inconveniente do *scraper* de detalhes, está no funcionamento da sua API. Ao contrário das demais APIs utilizadas até então, não é possível chamar a API de detalhes com apenas dados provenientes da lista de *inputs*, pois ela

exige de um *token* e um *cookie* que são apenas gerados durante o fluxo de acesso de páginas no Booking, caso contrário a requisição é bloqueada pelo site. Para solucionar esse problema, foi elaborado uma rotina de acesso que é executada no início de cada instância do *scraper*, de maneira a coletar esses dois metadados mencionados antes de começar a *scrapagem* de imóveis. A rotina funciona acessando a página inicial do Booking com a mesma sessão HTTP que será utilizada no restante da *scrapagem*, vinculando o *cookie* necessário à essa conexão. O *token* cobiçado pode ser achado dentro de um código JavaScript no HTML da página inicial. Com essa informação, é utilizado o BeatifulSoup para extrair o *token* da mesma resposta HTML retornada na requisição realizada para obter o *cookie*.

Mesmo assim, ainda falta mais uma informação para que o *scraper* de detalhes funcione: o ID individual de cada imóvel anunciado. Conforme explicado na Seção 4.1, o Booking permite o agrupamento de imóveis/quartos em um único anúncio, consequentemente, para realizar uma chamada à API de detalhes de imóveis, também é necessário enviar o ID do imóvel, o que não é uma informação que a Stays tem acesso. Esse obstáculo foi superado ao reparar que os IDs de imóveis nada mais são que o código do anúncio em que abrigam, só que com dois caracteres numéricos adicionados no final do ID. Esses números anexados ao código do anúncio representam o índice de cada imóvel e seguem uma lógica sequencial começando em “01” e indo até “99”, sem pular nenhum número. Com isso, o *scraper* de detalhes adquire os imóveis do anúncio consumindo começando pelo ID “código Booking + 01” e continua incrementando os dígitos adicionados no final do código até que seja retornada uma mensagem de invalidação, momento em que é parado a *scrapagem*, pois acabaram os imóveis do anúncio.

5.3.3 Informações coletadas

Após a extração dos dados, as informações coletadas são agrupadas com alguns metadados referentes à aquisição e inseridas em um dicionário (lista de dicionários para o caso dos *scrapers* de detalhes e comentários). Os metadados, informações obtidas (com as nomenclaturas que serão usadas nas tabelas), tipagem de dados e descrição do que cada informação representa, podem ser lidos nos apêndices: Apêndice A, Apêndice B, Apêndice C e Apêndice D.

5.4 IMPLEMENTAÇÃO DO *SCRIPT* DE REFIL DE INPUTS

O *script* de refil cumpre o papel de preencher as filas SQS com os dados de *inputs* que os *scrapers* precisam para operar corretamente. A lógica funcional do código é relativamente simples: primeiro é feito o download de todos os JSONs de um determinado diretório no S3, depois é feito o agrupamento e eliminação de duplicatas

(deduplicação) de todos os *inputs* baixados em uma única lista, por fim, a lista é enviada para a fila SQS selecionada, onde cada mensagem corresponde a um par de URL e ID de um único anúncio.

O download dos JSONs e o envio de mensagens ao SQS são feitos com o auxílio da biblioteca Boto3, com o envio de mensagens à fila sendo dividido entre *green threads* para acelerar o processo.

A funcionalidade de baixar todos os arquivos dentro de um repositório, ao invés de baixar apenas um objeto pela sua URI (Uniform Resource Identifier), foi implementada pensando na expansão do sistema. Os *inputs* podem começar a vir de fontes diferentes caso ocorra a expansão de *inputs*, portanto, serão armazenados em múltiplos objetos, por isso a implementação de um sistema capaz de baixar todos os arquivo de um diretório sem precisar a existência de cada objeto individual.

A seleção de qual diretório será usado para o *download* e para qual fila os dados serão enviados, é feita através de argumentos de linha de comando configuradas pela biblioteca Fire, permitindo que o mesmo código de refil possa ser usado em todos os *scrapers* do Booking. O procedimento de como e quando isso é feito ficará mais claro na próxima seção.

As filas SQS de *inputs* foram criadas manualmente pelo *console* da AWS e as configurações padrão colocadas podem ser vistas na Figura 18. Todas as filas criadas são do tipo FIFO, para não ocorrer casos de um anúncio ser processado mais de uma vez sem necessidade. Os parâmetros de maior relevância são o *default visibility timeout*, *message retention period* e *maximum message size*, que significam o tempo que uma mensagem tem para ser processada até que seja retornada à fila, o tempo de retenção que as mensagens tem na fila até que sejam deletadas e o tamanho máximo que uma mensagem pode ter, respectivamente.

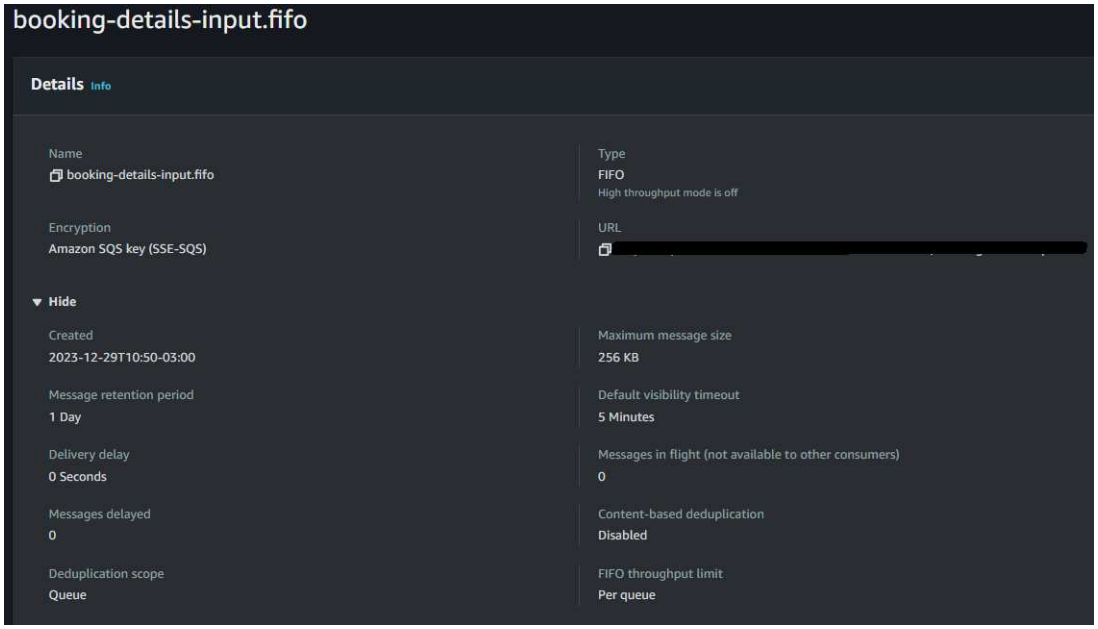
5.5 INTEGRAÇÃO DOS *SCRAPERS* LOCAIS COM A NUVEM

Esta etapa da metodologia tem como objetivo transformar o protótipo local dos *scrapers* em um programa autônomo que possa ser executado remotamente com monitoração mínima. Para isso, os protótipos serão adequados ao *framework* de execução em que o restante dos *scrapers* da Seazone operam e são conectados aos recursos de integração em nuvem.

5.5.1 *Framework* dos *scrapers*

O código dos *scrapers* é transformado em uma classe Python, que recebe os *inputs* de uma mensagem pelas variáveis internas do objeto instanciado. A lógica de busca e extração de informações é convertida em funções internas da classe. Também é implementado uma função pública designada para a execução automática

Figura 18 – Configurações da fila SQS de *inputs* do *scraper* de detalhes.



Fonte: Autor.

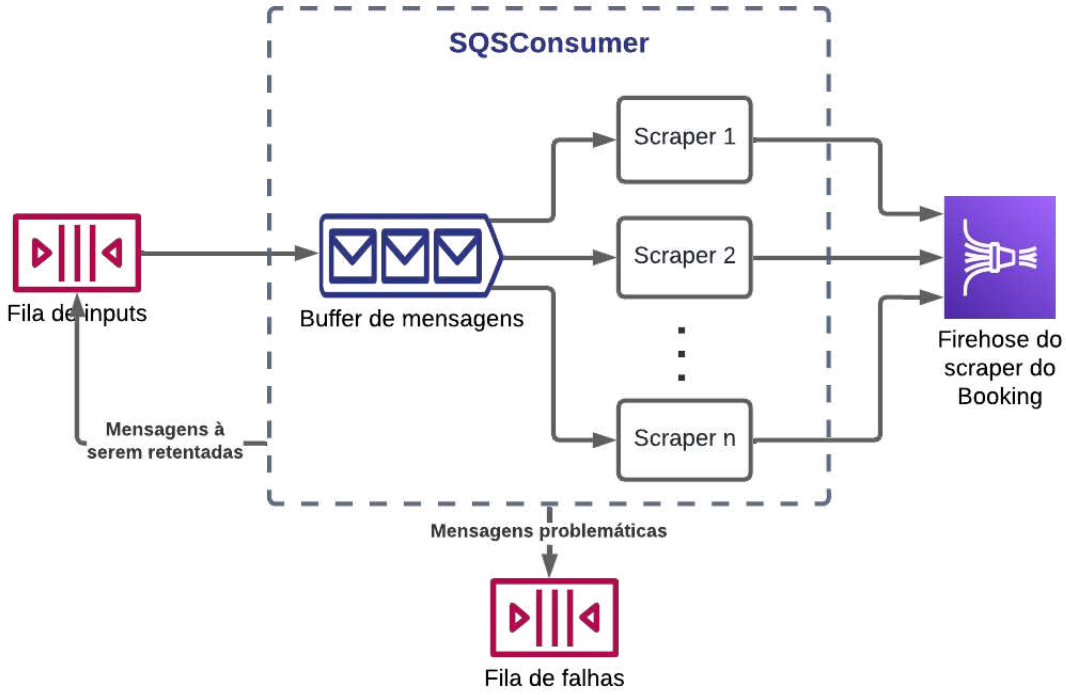
das funções internas. Essa função realiza o processo completo de web *scraping* para o anúncio passado nos argumentos de construção e manda as informações adquiridas para o Firehose.

As mensagens SQS são administradas por uma instância da classe SQSConsumer, da biblioteca Sqs.py. A instância constantemente consome as mensagens da fila e armazena elas em um *buffer* interno. Enquanto isso, é inicializado e mantido um número fixo configurável de instâncias da classe do *scraper*, com cada uma delas sendo responsável pela extração dos dados de um único anúncio por vez, seguindo a abordagem SIMD. Toda vez que um *scraper* não consegue coletar os dados do anúncio, a sua mensagem correspondente é enviada para o contexto do SQSConsumer, que incrementa o número de tentativas e decide se a mensagem é enviada de volta para a fila de *inputs* ou se ela vai para a fila de falhas. O procedimento descrito nesta seção, é representado pela Figura 19.

A paralelização interna dos *scrapers* por parte do SQSConsumer é feita através do uso das green threads, e só é efetiva por conta da natureza *I/O-bound* da tarefa de web *scraping*. Isso ocorre porque o limitador de velocidade do processo se encontra no tempo de espera por respostas do servidor do Booking, ou seja, o fato das *green threads* só serem capazes de usar um núcleo de processamento entre elas acaba não sendo um grande problema, pois a espera por retornos de requisições não necessita de recursos computacionais, permitindo que outros *scrapers* trabalhem enquanto isso.

Para evitar que *scrapers* (SQSConsumers) bloqueados pelo Booking continuem

Figura 19 – Fluxograma do funcionamento da classe SQSConsumer.



Fonte: Autor.

em operação, foi implementado uma lógica de contagem de requisições que foram retornadas com código 429 (código de status HTTP com significado de que a chamada foi bloqueada porque o cliente enviou requisições demais ao servidor), em que o *scraper* da um fim na execução do próprio código caso o contador ultrapasse um limite configurado. Dessa forma, quando os *scrapers* estiverem funcionando em ECS Fargate, as próprias tarefas ECS vão identificar que foram bloqueadas e encerrarão a si mesmas, fazendo com que o serviço ECS instancie novas tarefas com novos IPs no lugar das antigas. A lógica de contagem de requisições fica embutida na sessão HTTP personalizada que os *scrapers* do Booking usam.

5.5.2 Variáveis importantes

Códigos no *framework* dos web *scrapers* têm algumas de suas variáveis locais convertidas para variáveis de ambiente, que nada mais são que variáveis de configuração externas ao software que fornecem informações ou parâmetros que podem ser utilizados pelo programa durante sua execução. Essa conversão é feita principalmente para informações sensíveis ou cujo valor é importante, porém não inerente à lógica do código, como chaves, endereços de recursos e configurações de performance.

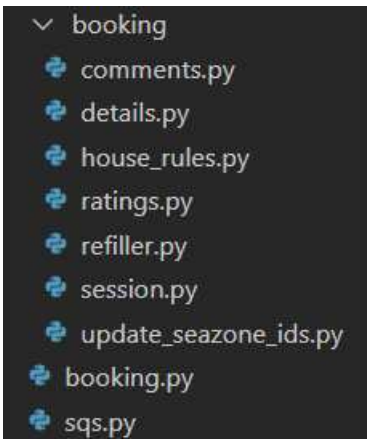
Enquanto o código é executado localmente, essas variáveis de ambiente ainda são acessadas de um arquivo cujo usuário tem acesso direto, porém quando o *scraper* começar a funcionar na ECS, elas passarão a ser importadas de um local seguro na AWS.

Uma variável de ambiente relevante para a otimização do sistema é a de concorrência de *scrapers*. Ela dita quantas *green threads* serão utilizadas na hora de paralelizar determinado *scraper*. A configuração ideal para essa variável é tentar deixá-la no maior valor possível, mas sem que os *scrapers* comecem a tomar muitos bloqueios de código 429.

5.5.3 Organização de arquivos

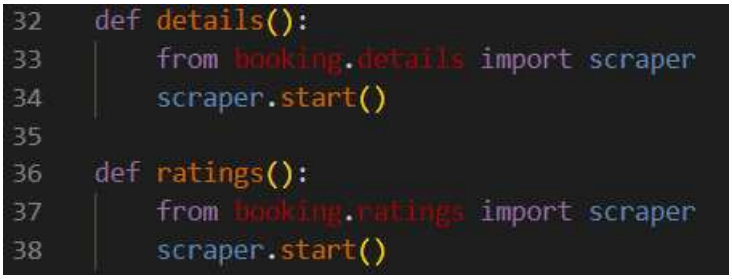
Os arquivos do projeto são organizados de acordo com a Figura 20. A pasta de nome “booking” contém o código fonte dos *scrapers* do Booking, do *scraper* da Stays (arquivo “update_seazone_ids.py”), do *script* de refil de filas e da sessão customizada que os *scrapers* usam para se comunicarem com o Booking. O arquivo “sqs.py” é literalmente a biblioteca Sqs.py citada algumas vezes ao longo da monografia. A execução dos *scripts* do projeto é realizada pelo arquivo “booking.py”, que serve como uma espécie de central de operação, onde cada código é representado por uma função que pode ser chamada diretamente pela CLI com o auxílio da biblioteca Fire. Para exemplificar o funcionamento deste *script*, a Figura 21 mostra um trecho de código do booking.py, pelo qual os *scrapers* de detalhes e avaliações podem ser executados pelos comandos “python booking.py details” e “python booking.py ratings”, respectivamente.

Figura 20 – Organização dos arquivos do projeto no repositório Pipe-scrapers.



Fonte: Autor.

Figura 21 – Trecho de código do *script* booking.py.



```
32  def details():
33      from booking.details import scraper
34      scraper.start()
35
36  def ratings():
37      from booking.ratings import scraper
38      scraper.start()
```

Fonte: Autor.

5.5.4 Criação de recursos na AWS

Para cada *scraper* do Booking, os seguintes recursos são manualmente criados pelo *console* da AWS: uma fila de falha SQS com as mesmas configurações da Figura 18, um diretório no *bucket* S3 da camada *raw* e um *crawler* configurado para escanear todas as partições do diretório do *scraper* sem agendamento.

Arquivos JSON com amostras de dados adquiridos localmente são temporariamente inseridos em cada um dos diretórios criados. Em seguida é feito a execução do *crawler* de cada *scraper* para que criem automaticamente as tabelas Glue com base na formatação dos dados da amostra. Desta forma, os Firehoses podem ser criados com conexão direta às tabelas Glue mesmo antes dos dados de verdade começarem a ser inseridos S3.

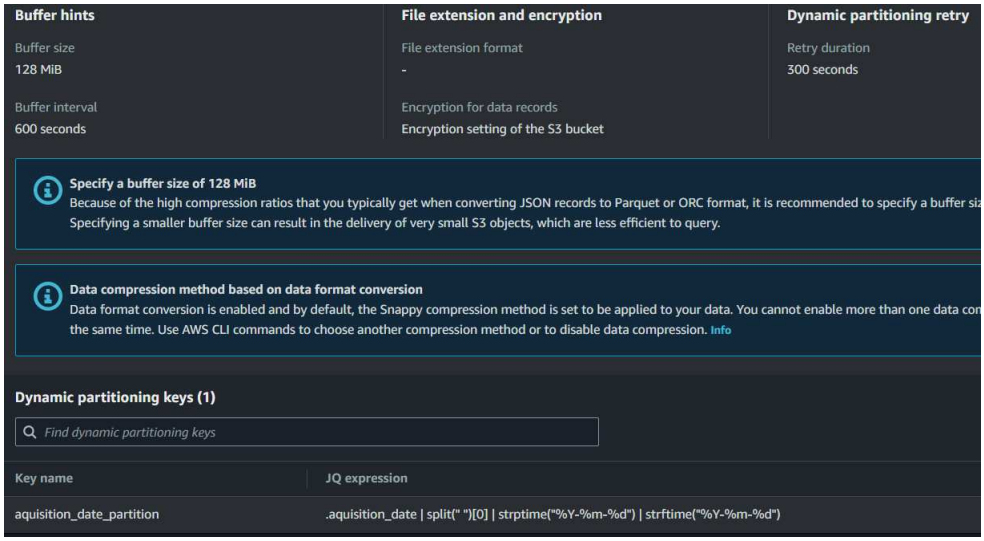
Por fim, são criados os Firehose Delivery Streams dos *scrapers*, cujas principais configurações podem ser visualizadas na Figura 22.

O particionamento dos dados é configurado por meio de uma JQ *expression* (expressões usadas para processar e transformar dados JSON, permitindo filtrar, extrair e modificar estruturas de dados), que extrai a data de valores na chave “aquisition_date” e a usa como partição. Dessa forma os dados obtidos pelos *scrapers* ficarão seccionados de acordo com a data em que foram adquiridos, tornando as consultas envolvendo essa coluna muito mais eficientes (é a coluna mais utilizada para filtragem de dados durante as análises mais frequentes da Seazone).

Os parâmetros *buffer size* e *buffer interval* representam a quantidade máxima de dados armazenados e o tempo máximo de retenção de dados no *buffer*, respectivamente. Quando qualquer uma dessas duas variáveis é ultrapassada, o Firehose insere os dados que acumulou no S3 na proporção de um Parquet por partição. Esse agrupamento de dados é importantíssimo para as consultas no Athena, pois reduz drasticamente a quantidade de arquivos à serem analisados, tornando-as mais baratas e eficientes.

Os Firehoses precisam que as tabelas Glue estejam sempre atualizadas com

Figura 22 – Captura de tela com as principais configurações dos Firehose Delivery Streams de *scrapers* do Booking.



Fonte: Autor.

os metadados corretos, pois eles utilizam essas tabelas para definir a esquematização dos dados de saída.

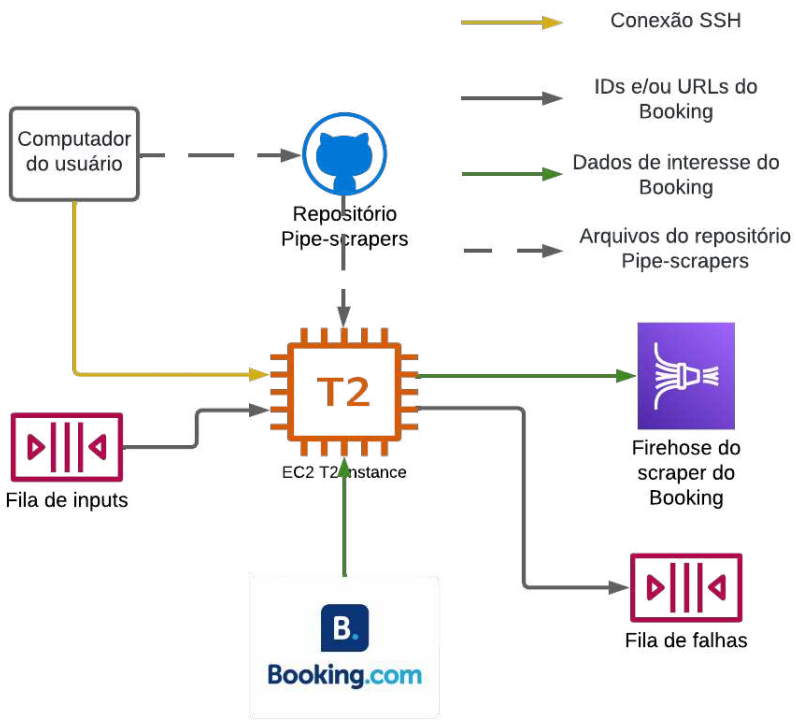
5.6 TESTE DO PROCESSO EM EC2

Nesta etapa é inicializada uma instância EC2 com baixa capacidade computacional para simular o comportamento dos *scrapers* na AWS em um ambiente controlado. Neste projeto foi utilizado uma instância do tipo T2.micro, que corresponde à 1 GiB de RAM e 1 vCPU (*virtual* Central Processing Unit) de processamento.

Após inicializar a instância EC2 e atualizar o repositório do Pipe-scrapers no GitHub com o código completo do projeto, é feito uma conexão SSH (Secure Shell) para acessar remotamente à instância, permitindo que o usuário utilize o terminal da máquina remota através do seu próprio terminal. Passado disso, a instância é configurada para ser capaz de executar o sistema completo de web *scraping* por conta própria, acessando o código fonte do projeto pelo GitHub do usuário, que irá simular a operação dos *scrapers* do Booking, conforme ilustrado na Figura 23.

O objetivo dos testes em EC2 é descobrir se os *scrapers* continuarão executando de forma adequada em ambiente AWS e, caso continuem, ajustar as configurações deles para que fiquem mais eficientes. Esta etapa pode parecer redundante considerando que os *scrapers* irão operar em ECS posteriormente, porém o ambiente EC2 permite a administração de testes bem mais dinâmicos do que em ECS, por conta do acesso remoto ao terminal.

Figura 23 – Fluxograma do funcionamento dos testes pontuais em EC2.



Fonte: Autor.

5.6.1 Resultado dos testes

Os testes começaram com execuções dos *scrapers* sem nenhum tipo de paralelismo (concorrência = 0) para checar se o Booking possui alguma espécie de barramento restrito aplicado em IPs da AWS (uma prática relativamente comum). Porém, nenhum dos *scrapers* sofreu nenhum tipo de bloqueio com esse nível de concorrência. Em seguida, foi realizada outra bateria de testes, porém com valores crescentes de concorrência. O maior valor atingido antes de qualquer barramento em qualquer um dos *scrapers* foi obtido com concorrência = 5 (equivalente a 6 *green threads* em operação), que acabou sendo o valor escolhido para ser passado para ambiente de produção. É provável que esse não seja o valor ótimo desta variável, pois o número de bloqueios se manteve baixo durante todos os testes, contudo, os *scrapers* já obtiveram uma eficiência satisfatória nesse nível de concorrência, então foi decidido manter esse valor para a primeira versão do sistema.

5.7 IMPLEMENTAÇÃO DO SISTEMA DE WEB *SCRAPING* EM ECS

Com a garantia de que o sistema está funcionando adequadamente em ambiente AWS, é feita a atualização da imagem Docker dos *scrapers* no ECR. Esse

processo é executado automaticamente ao criar um novo lançamento do ramo principal do repositório Pipe-scrapers no GitHub. Os detalhes do procedimento não são relevantes para o PFC, porém, de um ponto de vista geral, ele é executado por um fluxo de trabalho do GitHub Actions (plataforma de integração e entrega contínua do GitHub) que monta a imagem Docker a partir de um Dockerfile localizado dentro do repositório remoto.

Em seguida, são criadas as tarefas ECS das etapas do sistema de web *scraping*. No ECS, a cobrança é proporcional ao tamanho da instância (CPU e RAM) e ao tempo que ela fica em atividade. Por conta disso, é interessante configurar as tarefas para que utilizem o menor tamanho de máquina possível sem que percam a eficiência.

Para que o processo de web *scraping* seja paralelizável com o uso de múltiplas máquinas ECS, é interessante a criação de serviços ECS de gerenciamento de tarefas. Cada serviço ECS é atrelado à exatamente uma tarefa, portanto, mesmo que todas as tarefas utilizem as mesmas configurações e imagem Docker, ainda é necessário criar uma tarefa ECS por *scraper*. O contrário é verdade para o *script* de refil, pois ele não é um código que precisa ser paralelizado, permitindo que apenas uma tarefa seja criada para ele.

Cada *scraper* recebe sua própria tarefa ECS, em que as configurações relevantes são:

- **Sistema operacional da instância:** O Linux/x86_64 foi selecionado para manter o mesmo sistema operacional utilizado durante o desenvolvimento local dos *scrapers*;
- **Tamanho da instância provisionada:** A instância foi configurada com 0,25 vCPU de processamento e 0,5 GiB de memória RAM, que são os menores valores permitidos para as configurações. Isso é possível pelo fato dos *scrapers* consistirem de *scripts* extremamente leves, que conseguem ser executados com recursos mínimos;
- **Detalhes de contêiner:** Foi passada a URI da imagem Docker dos *scrapers* no ECR;
- **Local do arquivo de variáveis de ambiente:** Foi passado a URI do arquivo contendo todas as variáveis de ambiente do projeto;
- **Comando de CLI executado pelo contêiner:** Única diferença de configuração entre os *scrapers*. Foi passado o comando de execução individual de cada *scraper* no formato “python booking.py nome_do_scraper”.

A configuração da tarefa ECS do *script* de refil segue a mesma lógica do restante dos *scrapers*, com diferenças no comando de CLI de execução e tamanho da instância,

que foi configurada com 0,5 vCPU e 1 GiB de RAM, pois o *script* precisa de um pouco mais de memória para armazenar e deduplicar as mensagens (não é um problema agora, mas passará a ser quando o sistema for escalonado e aumentar o número de anúncios à serem coletados).

É criado um serviço ECS pra cada tarefa ECS de *scraper*. Existem duas configurações relevantes para essa etapa, com a primeira sendo a especificação de *deployment type*, que determina o método utilizado para implantar e atualizar as tarefas em execução. A opção de método escolhida é a *rolling update* que faz com que o ECS substitua gradualmente as tarefas problemáticas por novas, garantindo que a aplicação permaneça disponível durante o processo de atualização.

A segunda configuração relevante é a de que VPC e *subnets* o serviço vai utilizar. Aqui é escolhido a mesma VPC utilizada pelo restante dos *scrapers* da Seazone, pois ela permite o acesso à quatro *subnets* diferentes, cada uma possuindo um montante de 4096 IPs públicos à sua disposição. Essa configuração, aliada ao método *rolling update* de implantação, permitem o comportamento de constante aquisição de dados em larga escala, mesmo com ocasionais barramentos por parte do Booking, pois habilita a constante substituição de *scrapers* bloqueados por novas instâncias com IPs públicos diferentes.

5.8 AUTOMAÇÃO COMPLETA DO SISTEMA EM PRODUÇÃO

A automação completa do sistema de web *scraping* é realizada através da criação das máquinas de estado do Step Functions para cada processo de *scrapagem*, e da configuração e agendamento dos *crawlers* para a atualização constante dos metadados das tabelas Glue.

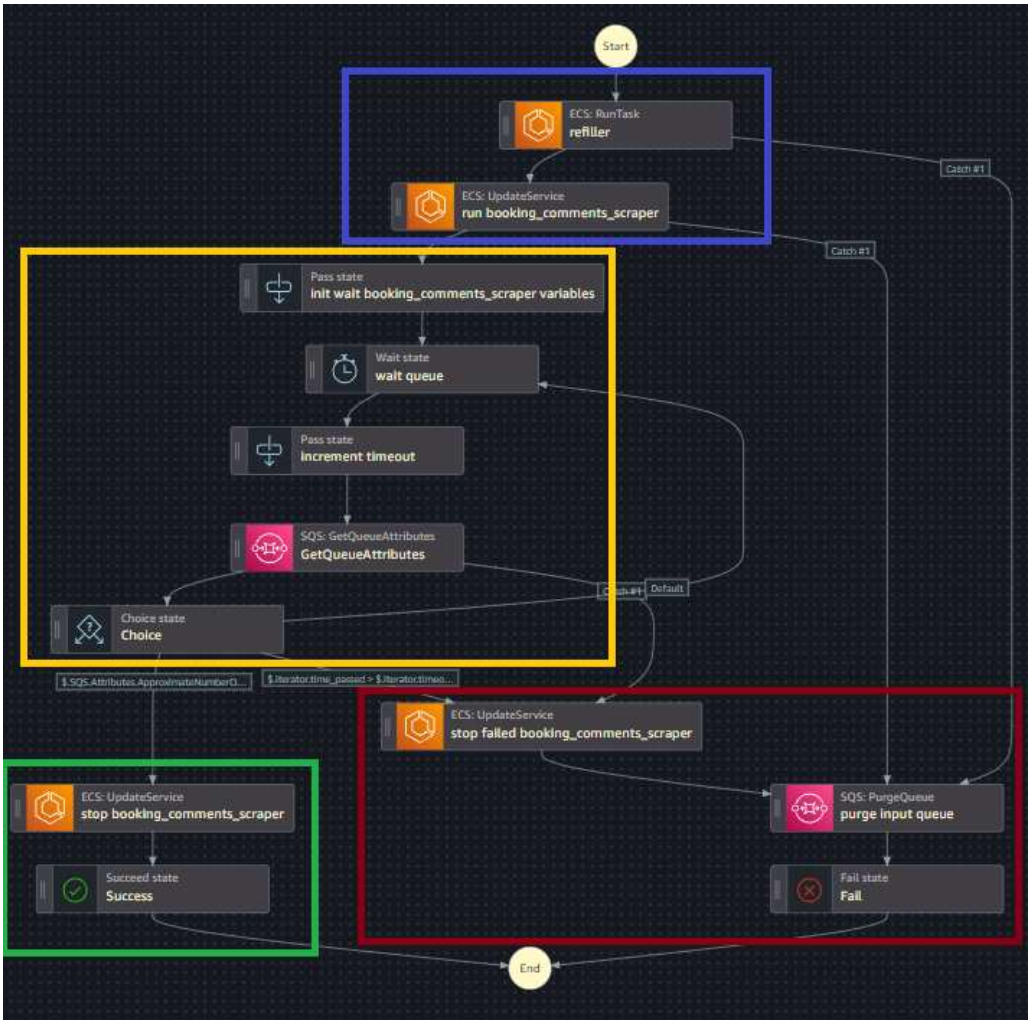
5.8.1 Configuração das máquinas de estado

A maquina de estado do Step Functions é um mecanismo visual de orquestração de processos com integrações nativas para quase qualquer serviço AWS. No contexto deste PFC, ele é usado principalmente para administrar as etapas do processo de web *scraping*. Cada *scraper* do Booking tem sua própria máquina de estados, porém todas elas seguem o mesmo padrão. A máquina de estados do *scraper* de comentários pode ser vista na Figura 24, que servirá como exemplo para a explicação da lógica de funcionamento geral.

Para uma melhor didática de explicação, a Figura 24 foi separada em quatro setores: inicialização (quadrado azul), espera (quadrado amarelo), fluxo de falha (quadrado vermelho) e fluxo de sucesso (quadrado verde).

O setor de inicialização começa na execução da tarefa ECS do *script* de refil de filas, que é chamado pelo estado “refiller” para preencher a fila do *scraper* de co-

Figura 24 – Máquina de estados do *scraper* de comentários.



Fonte: Autor.

mentários. O estado seguinte atualiza o serviço ECS do *scraper* para mudar o número alvo de tarefas em execução de 0 para um valor estático configurado. Atualmente esse número é 2, pois os *scrapers* do Booking ainda estão pegando apenas dados de listagens da Seazone, porém é a partir deste valor que o *scraper* consegue escalonar para quantidades maiores de instâncias operando em paralelo. Também foi adicionado uma clausula nos dois estados do setor de inicialização que aponta para o fluxo de falha caso de erro em algum estado.

A lógica de monitoramento da fila SQS de *inputs*, que identifica quando os *scrapers* devem ser parados, foi implementada no setor de espera. Isto é feito com o uso de três variáveis internas inicializadas no primeiro estado do setor: *timeout*, tempo de espera e tempo passado. Toda vez que é passado por cima do estado “wait queue” espera-se um número de segundos equivalente à variável tempo de espera. O estado “increment timeout” incrementa a variável tempo passado de acordo com o cálculo:

tempo passado + tempo de espera = novo tempo passado .

O estado de “GetQueueAttributes” é responsável por checar quantas mensagens a fila de *inputs* ainda possui e armazenar esse número em uma variável. Por fim, o estado de “Choice” checa duas condições: se a fila de *inputs* está ou não vazia e se a variável de tempo passado já ultrapassou a variável de *timeout*. Se a fila estiver vazia, o fluxo de sucesso é inicializado, caso a fila não estiver vazia e o tempo passado tiver ultrapassado o *timeout*, o sistema entra no fluxo de falha, e, se nenhum dos dois outros caminhos foi escolhido, o sistema volta para o estado de “wait queue” e recomeça o ciclo de espera. Com isso, tem-se implementado uma lógica de espera de esvaziamento da fila, com um esquema de *timeout*, para casos em que algo sai do controle.

Os fluxos de falha e sucesso têm o papel de reiniciar o sistema para seu estado inicial de zero mensagens na fila e zero instâncias em operação. As diferenças entre os dois está em o estado de sucesso não precisar esvaziar a fila de *inputs* explicitamente (pois ela já está vazia) e a classificação de falha ou sucesso na execução da máquina de estados.

Os web *scrapers* criados vão fazer parte do sistema semanal de aquisição de dados, e foram agendados para iniciar toda segunda-feira às 8h00 do horário de Brasília.

5.8.2 Configuração e agendamento de *crawlers*

Para que a automação do sistema esteja completa, é necessário configurar e sincronizar os *crawlers* para que tornem consultas às tabelas Athena um mecanismo consistente de disponibilização de dados. Isso pode ser feito com a simples criação de *crawlers* Glue configurados para explorar apenas as novas partições de dados (para evitar o processamento desnecessário de dados históricos) e agendando-os para que executem toda segunda, logo após o tempo de *timeout* dos *scrapers*, evitando possíveis complicações geradas pelos *crawlers* executando antes dos *scrapers*. Assim, as tabelas do sistema estarão sempre atualizadas com os dados mais frescos o possível.

5.9 IMPLEMENTAÇÃO DO *SCRAPER* DA STAYS

O *scraper* de URLs e IDs Booking da Stays consiste de um *scraper* mais simples, que não precisou seguir boa parte da metodologia de desenvolvimento, pois não tem a necessidade do uso dos mesmos mecanismos de paralelismo que os *scrapers* do Booking utilizam. Isso ocorre porque, independente da expansão do projeto para novas listas de anúncios, o *scraper* da Stays continuará servindo o mesmo propósito de

coletar os *inputs* de listagens Seazone. Por conta disso, mesmo que a empresa cresça rapidamente o seu portfolio de anúncios no Booking, o *scraper* da Stays continuará a cumprir seu papel adequadamente.

A estrutura lógica do *scraper* da Stays é similar à descrita na Subseção 5.3, funcionando a partir de uma lógica iterativa funcional sem paralelismo. Ele pode ser dividido em três partes funcionais: login, extração de dados e *upload* dos arquivos no S3.

Primeiramente, ele realiza o login na Stays a partir de uma requisição HTTP de autorização, utilizando as credenciais da sua conta na plataforma. O login e a senha são passados por meio de variáveis de ambiente e a conta acessada possui apenas as permissões necessárias para visualizar as páginas da seção de listagens do Booking.

Em seguida, é efetuado a coleta dos dados de interesse da seção de anúncios. O ID e URL das listagens podem ser achados no meio do HTML da aba individual de cada anúncio, portanto a extração de informações é feita através da biblioteca BeatifulSoup com o interpretador “lxml”. A paginação do site é realizada por manipulação de URL, adicionando o número da página no final da URL padrão.

Após coletar os IDs e URLs de todos os anúncios ativos da Seazone, o *scraper* faz o *upload* dos dados obtidos ao diretório de *inputs* de cada *scraper* do Booking no formato de arquivo JSON. Foi implementado uma lógica que escaneia os diretórios do *bucket* de *inputs*, listando os objetos com o mesmo nome do arquivo JSON gerado pelo *scraper*, lista que é utilizada para atualizar estes objetos com a versão atualizada dos *inputs*. Esta abordagem permite a integração de novos *scrapers* do Booking sem precisar de nenhuma configuração extra, basta criar um diretório novo e colocar manualmente o arquivo de *inputs* que ele já vai ser atualizado automaticamente.

Os únicos dois recursos que precisam ser criados para o funcionamento do *scraper* da Stays na AWS são: sua tarefa ECS e sua maquina de estados. As configurações da tarefa ECS são análogas as do restante dos *scrapers*, porém sem a necessidade de um serviço ECS de administração. Foi escolhido realizar o agendamento de execução pelo Step Functions apenas por motivos de padronização, porque a máquina de estados do *scraper* consiste de somente um estado de execução de tarefa ECS. O *scraper* foi inicialmente agendado para executar todo dia às 7h00 do horário de Brasília, pois há planos para a implementação de *scrapers* do Booking com frequência diária de aquisição.

6 ANÁLISE DOS RESULTADOS

Neste capítulo, é realizado a análise dos resultados obtidos com a implementação do sistema de aquisição de dados para o Booking.com. A Seção 6.1 avalia a conformidade do sistema com os requisitos funcionais estabelecidos, incluindo a análise de consistência dos dados coletados. A Seção 6.2 discute a conformidade com os requisitos não funcionais, abordando desempenho, escalabilidade, custo e centralização de códigos.

6.1 REQUISITOS FUNCIONAIS

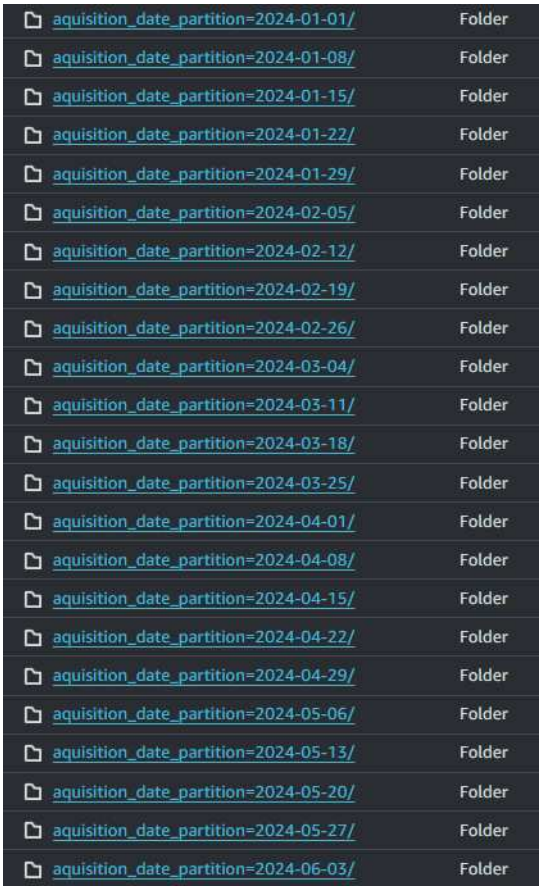
Nesta seção será feita a análise de adequação do projeto aos requisitos funcionais estipulados na Seção 3.5. Os requisitos incluem: extração de dados importantes, armazenamento dos dados, acessibilidade dos dados, abrangência e automação.

Com exceção do *scraper* de comentários, que teve sua primeira aquisição no dia 12 de fevereiro de 2024, todos os *scrapers* do Booking começaram a obter dados de forma completamente automática a partir do dia 1 de janeiro de 2024. O sistema operou normalmente do primeiro dia, até o momento de escrita desta seção (dia 6 de junho de 2024), totalizando 23 (17 para o *scraper* de comentários) semanas ininterruptas de aquisições em produção, exemplificado pela Figura 25, que mostra as partições geradas ao longo das semanas contendo dados da tabela de detalhes, localizadas no *bucket* da camada *raw* do *data lake* da Seazone. Com isso, pode-se considerar que os requisitos funcionais de automação e armazenamento de dados no *data lake* foram cumpridos com exito.

Graças ao sistema de metadados proporcionado pelo ambiente Glue (*crawlers*, tabelas e catalogo de dados), todos os dados coletados pelo sistema podem ser facilmente consultados por meio do AWS Athena, que é uma das principais ferramentas de análise de dados utilizada pela Seazone. Além disso, após a implementação dos *scrapers*, foram acoplados processos de disponibilização dos dados de detalhes, comentários, avaliações e regras da casa através de planilhas enviadas automaticamente por um canal no Slack da empresa, enfatizando a acessibilidade dos dados por usuários e processos. A Figura 26 mostra a captura de tela de um dos envios por Slack do processo citado.

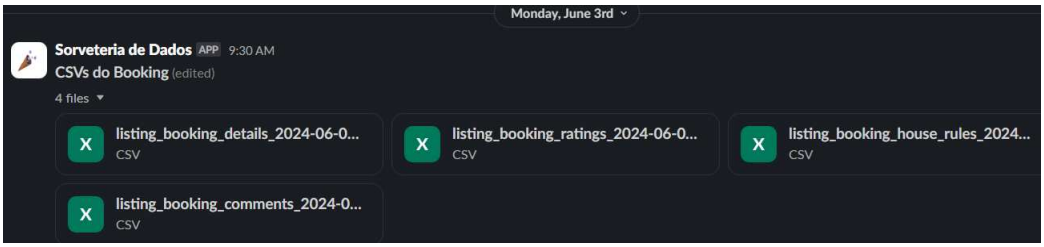
A abrangência dos *scrapers* a todas as listagens ativas da Seazone no Booking, é garantida pelo funcionamento do *scraper* da Stays, cuja execução automática diária vem acontecendo desde 13 de março de 2024, cumprindo o requisito de abrangência de web *scraping* aos anúncios ativos da Seazone no Booking.

Figura 25 – Captura de tela com as partições dos dados de detalhes.



Fonte: Autor.

Figura 26 – Captura do envio de dados pelo Slack da empresa.



Fonte: Autor.

6.1.1 Análises de consistência dos dados

Para assegurar que a extração de dados de relevância está ocorrendo da maneira correta, foram feitos alguns testes de sanidade em cima dos dados, garantindo que não esteja ocorrendo nenhum problema grande durante o processo de coleta. Os testes de sanidade são verificações simples e rápidas realizadas para garantir que um

sistema ou conjunto de dados esteja funcionando dentro dos parâmetros esperados antes de prosseguir para etapas mais detalhadas ou complexas de análise. Eles visam identificar problemas óbvios e falhas básicas que poderiam comprometer a validade dos resultados subsequentes. Por se tratar de um projeto de apenas coleta de dados, o uso de análises mais simples de checagem de qualidade dos dados já foi considerada suficiente, pois as verificações mais profundas serão realizadas ao longo do tempo, com o uso natural das informações por parte dos usuários.

A Tabela 2 apresenta algumas das métricas levantadas para a análise de integridade dos dados coletados. As análises foram realizadas a partir de consultas no Athena em cima dos dados *scrapados* de 1 de janeiro até 3 de junho de 2024.

Scraper	Porcentagem média de valores nulos	Duplicatas	Média de anúncios por semana
Detalhes	0,33%	0	693,4
Comentários	14%	0	390,6
Regras da casa	4,5%	0	693,4
Avaliações	34,92%	0	693,4

Tabela 2 – Análises de integridade dos dados adquiridos.

A média de anúncios por semana ficou exatamente igual para os *scrapers* de detalhes, regras da casa e avaliações, com 693,4 anúncios, número razoável considerando que o levantamento mais recente foi de 722 anúncios no Booking por parte da Seazone, significando um leve aumento no número total de listagens com o passar do ano. O número ter sido idêntico, diz respeito à consistência dos *scrapers*, que aparentam não estar cometendo erros imprevisíveis que acarretariam em uma variação neste número ao longo de diferentes *scrapers*

A quantidade média de anúncios menor em comentários, se dá ao fato desse *scraper* pegar os dados dos comentários em si, ou seja, caso uma listagem seja nova, que é o caso para boa parte dos anúncios da Seazone no Booking, ela ainda não vai possuir comentários e, portanto, não vai aparecer na tabela do *scraper*. Corroborando com essa análise, está o fato da aquisição mais recente de comentários ter vindo com 430 listagens diferentes, representando um aumento percentual significantemente maior que o experienciado pelo restante dos *scrapers*, explicado pelo ganho de comentários por parte dos anúncios ao longo do tempo.

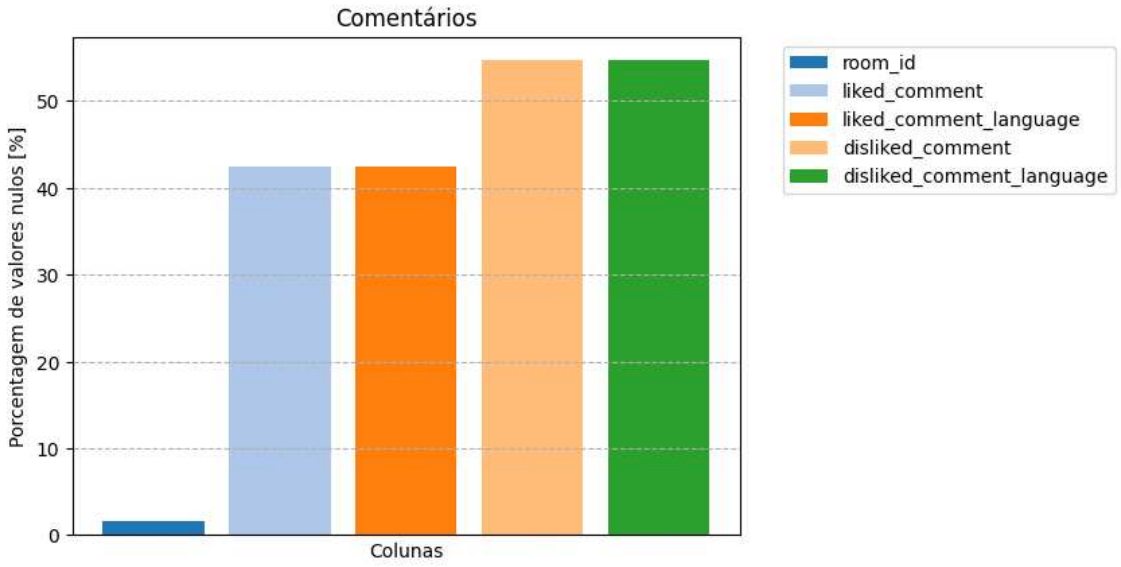
A coluna de duplicatas significa a quantidade de vezes a mais que um ID apareceu em uma única semana/aquisição. O ID de deduplicação utilizado em cada análise foi: ID do anúncio para avaliações e regras da casa, ID do comentário para comentários e ID do imóvel para detalhes. Ocorreram um total de zero duplicatas ao longo de todas as aquisições, significando que a lógica de administração de mensagens é funcional e nenhuma mensagem foi processada mais de uma vez sem necessidade.

A segunda coluna da Tabela 2 representa a média do percentual de valores nulos em cada uma das colunas. Para compreender melhor os resultados obtidos, foi

feito a análise individual das colunas de cada tabela. Começando pelo mais importante, nenhuma tabela apresentou valores nulos para IDs de deduplicação nem data de aquisição, que seriam casos problemáticos se tivessem ocorrido. Em detalhes, apenas uma coluna apresentou valores nulos, que foi a do tamanho do imóvel em metros quadrados em 3% de valores nulos. Foi checado manualmente alguns desses casos e todas as vezes que esse dado vinha nulo, era proveniente de um imóvel do anúncio que só era retornado por API, e não era visível pelo usuário.

A Figura 27 mostra as porcentagens de valores nulos individuais de cada coluna, com exceção de colunas com porcentagem de valores nulos inferiores à 0,5%. Os comentários do Booking são realizados junto com a avaliação da estadia, portanto é opcional escrever ou não um comentário, tanto que a coluna de avaliação veio com taxa de 0% de valores nulos. Os casos em que o ID do imóvel veio nulo que foram checados manualmente, realmente possuíam o ID do imóvel nulo no HTML da página.

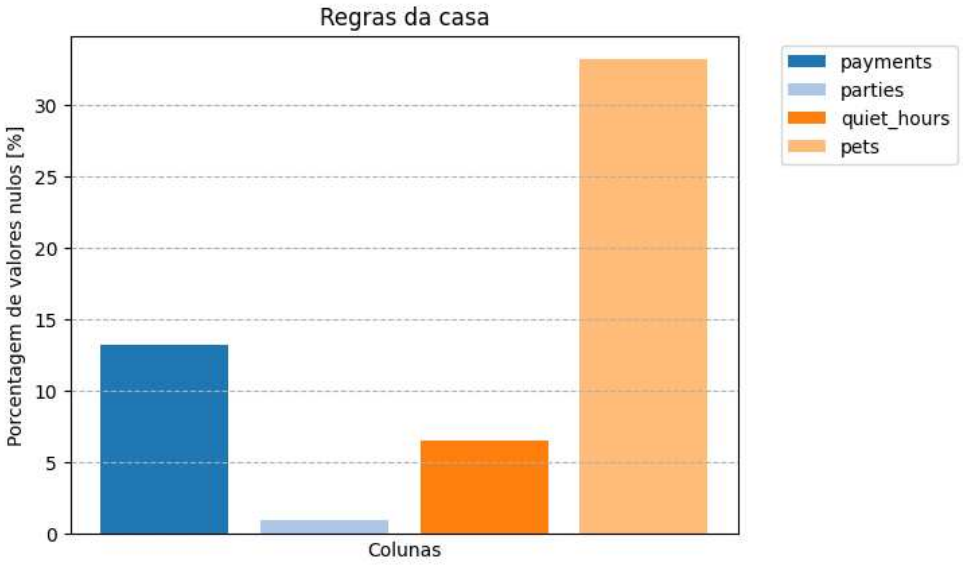
Figura 27 – Gráfico da porcentagem de valores nulos em colunas da tabela de comentários.



Fonte: Autor.

O gráfico com os valores da análise para a tabela de regras da casa pode ser visto na Figura 28. As regras da casa são campos completamente opcionais de serem colocados, portanto os valores nulos apresentados não causam problemas. Na realidade, todas as outras colunas referentes as regras da casa ficaram com uma porcentagem de valores nulos superiores à 0, porém inferiores à 0,5%, por isso não aparecem no gráfico. Os valores só são baixos desse jeito, pois os anúncios *scrapados* são todos da Seazone, que possui padrões específicos para determinadas regras da casa.

Figura 28 – Gráfico da porcentagem de valores nulos em colunas da tabela de regras da casa.



Fonte: Autor.

A Figura 29 apresenta os dados da análise de valores nulos de avaliações. Ao contrário dos comentários, quando um anúncio não possui avaliações ainda é inserido uma linha nova sobre ele na tabela explicando que ele tem zero avaliações. Isso faz com que os anúncios sem avaliações tenham valores nulos nas colunas de notas. A nota referente à avaliação de Wi-Fi gratuito apresenta uma porcentagem bem maior de valores nulos, pois só é avaliável em casos onde o imóvel oferece Wi-Fi gratuito.

Apesar do projeto não ter sido capaz de adquirir os dados de preço e disponibilidade, o restante dos dados importantes à serem extraídos estão sendo coletados com sucesso e aparentam ser consistentes com a realidade, dado as análises efetuadas.

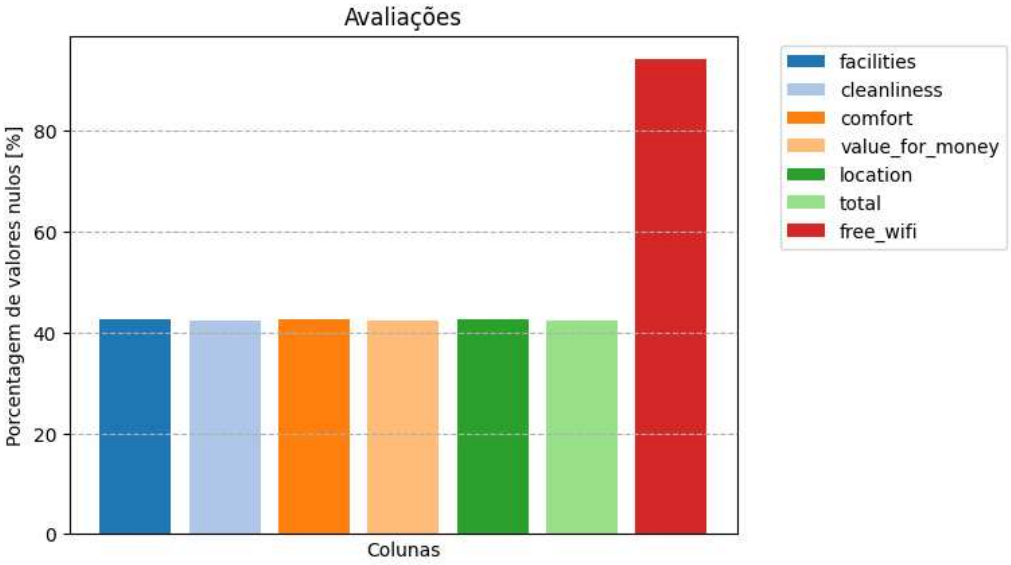
6.2 REQUISITOS NÃO FUNCIONAIS

Nesta seção será realizada a análise de conformidade do projeto com os requisitos funcionais estabelecidos na Seção 3.5. Os requisitos incluem: desempenho e escalabilidade, custo e centralização de códigos.

A centralização dos códigos foi cumprida por meio do uso do GitHub como repositório primário do projeto, facilitando a integração de terceiros em pontos futuros de expansão e manutenção do projeto. A integração direta do GitHub com o ECR e, por consequência, o ECS, promove fluidez no desenvolvimento de códigos a longo prazo, mesmo quando houver mais pessoas trabalhando em uma mesma funcionalidade.

Os requisitos de custo, desempenho e escalonabilidade estão fortemente interli-

Figura 29 – Gráfico da porcentagem de valores nulos em colunas da tabela de avaliações.



Fonte: Autor.

gados, pois ao se tratar de infraestrutura em nuvem, principalmente utilizando recursos *serverless*, o desempenho acaba sendo muitas vezes diretamente proporcional ao custo. Seguindo a mesma linha de raciocínio, não dá para falar de escalonabilidade sem levar em conta os custos, pois no ambiente da AWS, quase sempre é possível escalonar um serviço para dimensões astronômicas, e o limitador acaba sendo o preço de tal façanha.

O maior custo do projeto sem sombra de dúvidas é o ECS, e o restante dos recursos acabam sendo muito baratos em em contraste. Para decidir se um *scraper* tem ou não um bom desempenho, será feito uma comparação direta com o *scraper* de detalhes de anúncios do Airbnb, cujo desempenho é suficiente para permitir a empresa a usá-lo uma vez por semana em aproximadamente 550 mil imóveis do Airbnb no Brasil. A eficiência dos *scrapers* será mensurado pela quantidade de anúncios que cada instância consegue processar por unidade de tempo, descrita pela formula:

$$\frac{\text{total de anúncios processados}}{\text{instâncias ECS em paralelo} \cdot \text{tempo em horas}} = \text{desempenho}\ .$$

Por exemplo: o *scraper* de detalhes do Booking coleta as informações de 723 anúncios em 27 minutos com duas instâncias ECS em paralelo, portanto a sua eficiência é de:

$$\frac{723}{2 \cdot 27/60} = 803,3 \frac{\text{anúncio}}{\text{instância} \cdot h}\ .$$

Na Tabela 3, é feita a comparação de desempenho entre os *scrapers* do Booking e o *scraper* de detalhes do Airbnb.

Scraper	Eficiência [anúncio/(instância·*h*)]	Comparação com o *scraper* de detalhes Airbnb
Detalhes Airbnb	1309,5	1
Detalhes Booking	803,3	0,61
Comentários	2169	1,66
Regras da casa	903,8	0,69
Avaliações	1008,9	0,77

Tabela 3 – Tabela de comparação de eficiência entre os *scrapers* do Booking e o de detalhes do Airbnb.

Apesar da maioria dos *scrapers* ter uma eficiência menor que a do *scraper* de detalhes do Airbnb, ainda sim é mais que o suficiente para o escalonamento do número de listagens adquiridas, considerando que o próximo passo seria muito provavelmente a aquisição de listagens em pontos estratégicos para a Seazone, não o Brasil inteiro como é feito em detalhes do Airbnb. Ainda por cima, ao contrário do Airbnb, cada anúncio no Booking pode (e muitas vezes tem) vários imóveis listados em um único anúncio, o que aumentária consideravelmente a eficiência de todos os *scrapers* do Booking caso for considerado a eficiência por imóvel ao invés de por anúncio.

Conforme posto anteriormente, os custos em ECS nesse tipo de projeto são bem mais prominentes do que o restante. O método de cobrança do ECS é feito de acordo com o tamanho da instância e o tempo de execução. As instâncias utilizadas no *scraper* de detalhes do Airbnb tem as mesmas dimensões que as usadas nos *scrapers* do Booking, portanto a comparação de desempenho também acaba sendo uma comparação direta de custo. Com isso, é possível afirmar que o custo benefício do projeto cumpre os requisitos estabelecidos.

7 CONSIDERAÇÕES FINAIS

A implementação do sistema de web *scraping* para aquisição de dados do Booking pela Seazone, representa um avanço importante no gerenciamento e otimização das operações de aluguel por temporada da empresa. Desde a concepção até a implementação final, o projeto abordou uma série de desafios técnicos e funcionais, envolvendo conceitos relevantes da área de engenharia de dados.

Inicialmente, a integração com o Booking revelou a necessidade de mais um sistema de aquisição de dados automatizado, dada a ausência de dados externos e internos detalhados. A partir dessa premissa, foi desenvolvida uma metodologia sistemática de desenvolvimento dos *scrapers* baseada em experiências passadas, garantindo que cada etapa fosse executada com um propósito claro no contexto geral do projeto. A exploração detalhada das páginas do Booking, utilizando ferramentas simples como o Developer Tools, permitiu identificar e validar os dados que poderiam ser coletados, estabelecendo uma base sólida para o desenvolvimento subsequente dos *scrapers*.

Infelizmente não foi possível chegar em uma boa solução para o problema de coleta das informações de preço e disponibilidade dos imóveis, que trariam grande valor para o sistema. Mesmo assim, o desbravamento minucioso da plataforma permitiu que os *scrapers* fossem projetados de maneira a utilizar abordagens similares em seu funcionamento, sem a necessidade de implementação de mecanismos mais complexos, porém menos eficientes, de web *scraping*, como a simulação de navegadores.

A integração dos *scrapers* com a nuvem provou-se uma das etapas mais desafiadoras do PFC, necessitando o entendimento do funcionamento de diversos recursos proporcionados pela AWS e de conceitos de computação concorrente, além de como fazer com que as diferentes partes da arquitetura se encaixem e formem um sistema coeso.

A análise dos resultados foi satisfatória, cumprindo os requisitos técnicos estabelecidos, com as informações relevantes de todas as listagens ativas da Seazone no Booking sendo coletadas de forma automática, contínua e integrada com o restante dos processos de dados da Seazone. Em resumo, o projeto de desenvolvimento dos *scrapers* para o Booking atingiu seus objetivos principais, estabelecendo uma infraestrutura de coleta de dados eficiente e escalável que permitirá à Seazone tomar decisões mais informadas e competitivas.

A base de desenvolvimento construída neste PFC também abriu diversas oportunidades de continuação do projeto, afinal de contas, apenas o início do sistema de web *scraping* do Booking foi construído. Um caminho claro de expansão do projeto seria a integração de listagens externas à Seazone no sistema, permitindo as análises baseadas em concorrentes que já era uma das motivações claras do PFC.

Outra perspectiva futura está na aquisição de diferentes tipos de dados da plataforma. O Booking proporciona uma série de informações interessantes para a área de análise de mercado, como *tags* de qualidade do anúncio e de parceria com a plataforma. Mesmo assim, a perspectiva mais importante em termos de coleta de dados novos, seria a continuação das tentativas de extrair os dados de preço e disponibilidade da plataforma, pois são de grande importância para uma série de processos da empresa, abrindo uma outra dimensão de análise dos imóveis do Booking.

Por fim, a otimização de processos já existentes nunca é uma abordagem ruim. Ideias como: baterias de simulações de *scrapers* com variação dos parâmetros de performance para tentar deixá-los mais eficientes, ou, experimentos com novas tecnologias em nuvem que possam trazer benefícios de uso diferentes, podem acabar melhorando ainda mais o sistema já estabelecido e possivelmente abrindo espaço para novas perspectivas de expansão.

REFERÊNCIAS

ALICE, Rey; MICHAEL, Freitag; THOMAS, Neumann. Seamless Integration of Parquet Files into Data Processing. **Journal of Data Processing and Integration**, v. 45, n. 2, p. 123–135, 2023. DOI: 10.1002/jdpi.2023.12345.

AMAZON WEB SERVICES, Inc. **Boto3 Documentation**. [*S.l.*: *s.n.*]. Acesso em: 2024-05-24. Disponível em: https://boto3.amazonaws.com/v1/documentation/api/latest/index.html.

AMAZON WEB SERVICES, Inc. **What is Amazon Athena?** [*S.l.*: *s.n.*]. Acesso em: 2024-05-23. Disponível em: https://docs.aws.amazon.com/athena/latest/ug/what-is.html.

AMAZON WEB SERVICES, Inc. **What is Amazon EC2?** [*S.l.*: *s.n.*]. Acesso em: 2024-05-23. Disponível em: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html.

AMAZON WEB SERVICES, Inc. **What is Amazon Elastic Container Registry (ECR)?** [*S.l.*: *s.n.*]. Acesso em: 2024-05-23. Disponível em: https://docs.aws.amazon.com/AmazonECR/latest/userguide/what-is-ecr.html.

AMAZON WEB SERVICES, Inc. **What is Amazon Elastic Container Service (ECS)?** [*S.l.*: *s.n.*]. Acesso em: 2024-05-23. Disponível em: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html.

AMAZON WEB SERVICES, Inc. **What is Amazon S3?** [*S.l.*: *s.n.*]. Acesso em: 2024-05-23. Disponível em: https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html.

AMAZON WEB SERVICES, Inc. **What is Amazon Simple Queue Service (SQS)?** [*S.l.*: *s.n.*]. Acesso em: 2024-05-23. Disponível em: https://docs.aws.amazon.com/AWSSimpleQueueService/latest/SQSDeveloperGuide/welcome.html.

AMAZON WEB SERVICES, Inc. **What is Amazon VPC?** [*S.l.*: *s.n.*]. Acesso em: 2024-05-23. Disponível em: https://docs.aws.amazon.com/vpc/latest/userguide/what-is-amazon-vpc.html.

AMAZON WEB SERVICES, Inc. **What is AWS Glue?** [*S.l.*: *s.n.*]. Acesso em: 2024-05-23. Disponível em: https://docs.aws.amazon.com/glue/latest/dg/what-is-glue.html.

AMAZON WEB SERVICES, Inc. **What is AWS? - Amazon Web Services (AWS)**. Acesso em: 2024-05-23. Disponível em: https://aws.amazon.com/pt/what-is-aws/.

BASHARI, Rad Babak; JOHN, Bhatti Harrison; MOHAMMAD, Ahmadi. An introduction to docker and analysis of its performance. **International Journal of Computer Science and Network Security (IJCSNS)**, v. 17, n. 3, p. 228, 2017.

BOOKING.COM. **Making it easier for everyone to experience the world**. [Acesso em: 15/05/2024]. Booking Holdings Inc. Disponível em: https://about.booking.com/.

IBM. **Docker**. Acesso em: 2024-05-23. Disponível em: https://www.ibm.com/topics/docker.

IMPERVA. **2024 Bad Bot Report**. [*S.l.*: *s.n.*], 2024. Acesso em: 2024-05-20. Disponível em: https://www.imperva.com/resources/resource-library/reports/2024-bad-bot-report/.

KHDER, Moaiad Ahmad. Web Scraping or Web Crawling: State of Art, Techniques, Approaches and Application. **Int. J. Advance Soft Compu. Appl**, v. 13, n. 3, p. 144–160, 2021. DOI: 10.15849/IJASCA.211128.11.

LASKOWSKI, Nicole. **Data lake governance: A big data do or die**. Acesso em: 2024-05-21. 2016. Disponível em: https://www.techtarget.com/searchcio/feature/Data-lake-governance-A-big-data-do-or-die.

MARTIN-FUENTES, Eva; MELLINAS, Juan Pedro. Hotels that most rely on Booking.com–online travel agencies (OTAs) and hotel distribution channels. **Tourism Review**, v. 73, n. 4, p. 465–479, 2018.

MILOSLAVSKAYA, Natalia; TOLSTOY, Alexander. Big data, fast data and data lake concepts. **Procedia Computer Science**, Elsevier, v. 88, p. 300–305, 2016.

PHOCUSWRIGHT. **By 2023, Hotel Supplier Direct Will Again Surpass OTAs**. Acesso em: 2024-05-25. 2022. Disponível em:

https://www.phocuswright.com/Travel-Research/Research-Updates/2022/by-2023-hotel-supplier-direct-will-again-surpass-otas.

RICHARDSON, Leonard. **Beautiful Soup Documentation**. [*S.l.*: *s.n.*]. Acesso em: 2024-05-24. Disponível em: https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

SEVERANCE, Charles. Discovering JavaScript Object Notation. **Computer**, v. 45, n. 4, p. 6–8, 2012. DOI: 10.1109/MC.2012.132.

SIRISURIYA, SCM de S. A Comparative Study on Web Scraping. **Proceedings of 8th International Research Conference, KDU**, General Sir John Kotelawala Defence University, p. 134–140, 2015.

SRUJAN. **Parallel Computing**. [*S.l.*: *s.n.*], 2023. Acesso em: 2024-05-22. Disponível em: https://medium.com/@srujan.pat2004/parallel-computing-d888a2b6056e.

STEDMAN, Craig; LUTKEVICH, Ben. **Data Lake Definition**. Acesso em: 2024-05-21. Disponível em: https://www.techtarget.com/searchdatamanagement/definition/data-lake.

APÊNDICE A – DADOS DO *SCRAPER* DE DETALHES

Dados adquiridos pelo *scraper* de detalhes:

- **hotel_id (*string*):** ID interno do anúncio;
- **room_id (*string*):** ID interno do imóvel;
- **hotel_name (*string*):** Título do anúncio;
- **city_name (*string*):** Nome da cidade em que se localiza o anúncio;
- **accommodation_type (*string*):** Tipo de acomodação do imóvel (apartamento, casa, etc.);
- **room_surface_in_m2 (*integer*):** Tamanho do imóvel em métros quadrados;
- **number_of_bathrooms (*integer*):** Número de banheiros no imóvel;
- **number_of_rooms (*integer*):** Número de cômodos no imóvel;
- **room_facilities (*array*):** Lista de comodidades do imóvel;
- **aquisition_date (*string*):** *String* da data e hora da aquisição dos dados.

APÊNDICE B – DADOS DO *SCRAPER* DE COMENTÁRIOS

Dados adquiridos pelo *scraper* de comentários:

- **hotel_id (*string*):** ID interno do anúncio;
- **room_id (*string*):** ID interno do imóvel em que o hóspede ficou;
- **comment_id (*string*):** ID interno do comentário;
- **rating (*integer*):** Avaliação de estadia atribuída pelo hóspede;
- **liked_comment (*string*):** Comentário positivo escrito pelo hóspede;
- **liked_comment_language (*string*):** Lingua em que o comentário positivo foi escrito;
- **disliked_comment (*string*):** Comentário negativo escrito pelo hóspede;
- **disliked_comment_language (*string*):** Lingua em que o comentário negativo foi escrito;
- **commenter_nationality (*string*):** Nacionalidade do hóspede;
- **stay_length (*integer*):** Duração da estadia do hóspede;
- **date (*string*):** *String* da data do comentário;
- **aquisition_date (*string*):** *String* da data e hora da aquisição dos dados.

APÊNDICE C – DADOS DO *SCRAPER* DE AVALIAÇÕES

Dados adquiridos pelo *scraper* de avaliações:

- **hotel_id (*string*):** ID interno do anúncio;
- **facilities (*float*):** Nota média da avaliação de comodidades do anúncio;
- **cleanliness (*float*):** Nota média da avaliação de limpeza do anúncio;
- **comfort (*float*):** Nota média da avaliação de conforto do anúncio;
- **value_for_money (*float*):** Nota média da avaliação de custo-benefício do anúncio;
- **location (*float*):** Nota média da avaliação de localização do anúncio;
- **total (*float*):** Nota média geral das avaliações do anúncio;
- **free_wifi (*float*):** Nota média da avaliação de Wi-Fi gratuito do anúncio;
- **number_of_ratings (*integer*):** Número total de avaliações do anúncio;
- **aquisition_date (*string*):** *String* da data e hora da aquisição dos dados.

APÊNDICE D – DADOS DO *SCRAPER* DE REGRAS DA CASA

Dados adquiridos pelo *scraper* de regras da casa:

- **hotel_id (*string*):** ID interno do anúncio;
- **checkin (*string*):** Regra de *checkin* imposta pelo anunciante;
- **checkout (*string*):** Regra de *checkout* imposta pelo anunciante;
- **cancellation (*string*):** Regra de cancelamento imposta pelo anunciante;
- **children (*string*):** Regra de presença de crianças imposta pelo anunciante;
- **age_restriction (*string*):** Regra de idade mínima imposta pelo anunciante;
- **payments (*string*):** Regra de formas de pagamento imposta pelo anunciante;
- **parties (*string*):** Regra de festas imposta pelo anunciante;
- **quiet_hours (*string*):** Regra de horário de silêncio imposta pelo anunciante;
- **pets (*string*):** Regra de horário de presença de *pets* imposta pelo anunciante;
- **fine_print (*string*):** Pormenores do anúncio;
- **aquisition_date (*string*):** *String* da data e hora da aquisição dos dados.