Universidade Federal do Paraná
Setor de Ciências Exatas
Departamento de Estatística
Programa de Especialização em *Data Science* e *Big Data*

Gabriel Yuri Silva Ribeiro

Plataformas de web scraping em ambiente de computação em nuvem: Uma perspectiva prática

Curitiba
2022

Gabriel Yuri Silva Ribeiro

Plataformas de web scraping em ambiente de computação em nuvem: Uma perspectiva prática

Monografia apresentada ao Programa de Especialização em Data Science e Big Data da Universidade Federal do Paraná como requisito parcial para a obtenção do grau de especialista.

Orientador: Luis C. E. Bona

Curitiba
2022

Especialização em Data Science e Big Data
Universidade Federal do Paraná
dsbd.leg.ufpr.br

UFPR
UNIVERSIDADE FEDERAL DO PARANÁ

Plataformas de web scraping em ambiente de computação em nuvem: Uma perspectiva prática

Gabriel Yuri Silva Ribeiro[1]
Pedro Augusto de Lima e Silva[2]
Luis C. E. Bona[3]

Resumo

Este trabalho tem como enfoque apresentar uma solução de engenharia de dados para o problema de scraping de dados web de maneira sistêmica e orquestrada, utilizando de uma infraestrutura hospedada em nuvem e declarada como código, bem como um pipeline de integração contínua e deploy contínuo para operacionalizar a adição de novos recursos e funções. Desta forma, um website com informações de partidas competitivas do jogo Counter Strike: Global Offensive foi escolhido como fonte de dados, que forneceu arquivos de partidas que foram tratados até a criação de tabelas em um data warehouse. Por fim, uma breve proposta de utilização dos dados é apresentada.
Palavras-chave: Computação em nuvem, engenharia de dados, pipelines, web scraping

Abstract

This work focuses on presenting a data engineering solution to the problem of web data scraping in a systemic and orchestrated way, using a cloud-hosted infrastructure and declared as code, as well as a continuous integration pipeline and continuous deployment to smoothen the addition of new features and functions. In this way, a website with information from competitive matches of the game Counter Strike: Global Offensive was chosen as a source of data, which provided match-related files to be then processed to the creation of tables in a data warehouse. Finally, a brief proposal for the use of the data is presented.
Keywords: *Cloud computing, data engineering, pipelines, web scraping*

1 Introdução

A crescente demanda por análises e modelos estatísticos tem evidenciado muito o protagonismo de cientistas de dados, estatísticos e analistas de dados, especialmente após 2010 [1]. E para que toda essa cadeia de consumidores prospere, existe a urgente necessidade por dados confiáveis, facilmente trabalháveis e relevantes.

A partir dessa demanda surgiu o engenheiro de dados. Essa profissão se compromete de arquitetar, construir e monitorar pipelines de dados, que têm como função atualizar data lakes ou data warehouses com informações frescas para que os analistas montem seus dashboards e cientistas montem seus modelos e análises [1].

Atualmente, existem diversas fontes de dados - sistemas internos de empresas, portais públicos, consultorias especializadas em inteligência, websites, Application Programming Interfaces (APIs) e muitas outras. Cada uma dessas fontes pode fornecer dados em inúmeras formas e tamanhos, como .CSV, .JSON, .PARQUET, .AVRO e até .JPG [2].

Plataformas atuais de dados são capazes não só de lidar com dados tabulados e bem comportados, mas também com dados nas suas diversas formas, sendo capazes de tratar dados de diferentes fontes de informação em tabelas claras e bem estruturadas, capazes de alimentar relatórios, alertas, modelos e estudos.

Por conta de ser um tema ainda novo e pouco explorado e com potencial para muitos projetos de dados, o objeto de estudo escolhido foram partidas do jogo CS:GO, pois também oferecem o desafio extra de se trabalhar com dados não estruturados. Arquivos .dem possuem tamanhos de 40 a 250 MB.

Para que seja possível coletarmos dados de forma massiva de websites relacionados ao assunto, a solução deve ser sistêmica e automatizada. É nesse contexto que um webscraper pode ser utilizado para realizar essa ingestão [3]. Um código de webscraping pode extrair dados de um website utilizando recursos como a linguagem de marcação de hipertexto (HTML) da página, interfaces de programação aplicações (APIs)e até mesmo da conexão da rede [3]. A implementação escolhida se baseia em encontrar os identificadores de partidas competitivas, para fazer o download de arquivos de replay (.dem), que serão salvos de modo a serem utilizados posteriormente para a criação de produtos de dados.

No caso de dados provenientes de partidas do jogo CS:GO, muitos produtos de dados podem ser construidos: Estatísticas gerais de jogadores em um campeonato, estudos de distribuição da posição dos jogadores

[1] Aluno do programa de Especialização em Data Science & Big Data, gyribeiro2014@gmail.com.
[2] Analista de dados da Gamersclub, pdrals16@gmail.com.
[3] Professor do Departamento de Informática - DInf/UFPR.

em cada um dos mapas, modelos preditores de vitória/derrota, ranqueamento de habilidade e muitas outras possibilidades.

O ciclo de vida desses dados é um aspecto importante do ponto de vista de engenharia, onde vários estágios são necessários para que os produtos de dados e análises sejam possíveis. As principais etapas são as 5 listadas abaixo:

- Geração
- Armazenamento
- Ingestão
- Transformação
- Disponibilização

Além disso, muitos outros aspectos complementares estão intrinsecamente ligados a todas as etapas do ciclo são cruciais para garantir uma boa segurança, manutenibilidade e evolução do pipeline.

A seguir, um típico ciclo de vida no contexto de engenharia de dados e suas nuâncias estão exemplificados.

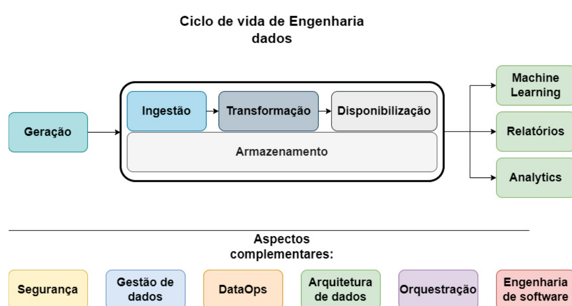

Figura 1: Visão holística do ciclo de vida de Engenharia de dados. [1]

De modo a conectar a ponta de criação dos dados com o seu consumo final, é necessário criar um pipeline de dados. Esse pipeline deve conter os passos de extração, tratamento e armazenamento dos dados, de modo a habilitar o consumidor final a fazer suas análises através de apenas uma consulta no data warehouse.

Existem diversas ferramentas que se comprometem de executar tarefas sequencialmente, onde frequentemente o resultado de saída de uma tarefa deve compor os parâmetros de entrada da entrada da próxima. No caso do crawler, uma tarefa que procura por identificadores de partidas deverá passar esses identificadores para que sejam processados na próxima etapa, e assim sucessivamente. Alguns dos orquestradores de tarefas mais comuns são o Apache Airflow, AWS Step Functions, Databricks Jobs, entre outros. Esses orquestradores não realizam a computação em si, mas ordenam a instâncias de computação que executem cada tarefa na sequência adequada [4].

Todas as tarefas dentro do Apache Airflow podem ser escritas em uma gama de linguagens, e naturalmente a linguagem em Python é a mais comum, dado que o próprio Apache Airflow é um projeto open-source com seu código escrito em Python. Escrever as funções e tarefas em Python possibilita o uso nativo de métodos já desenvolvidos pela comunidade, como por exemplo a comunicação com serviços da AWS, com toda a autenticação já encapsulada em funções nativas do Airflow. Para escrever o código com versionamento, algo necessário para desempenhar as tarefas citadas acima, uma ferramenta essencial é o git.

O git se tornou o padrão mundial para o controle de versão de maneira distribuída, sendo muito popular em times de programação. Estes repositórios fazem com que seja fácil trabalhar offline ou remotamente, e permitem que diversos contribuidores escrevam código para o mesmo projeto. Numa aplicação típica, os desenvolvedores comprometem seu trabalho localmente, e então sincronizam sua cópia do repositório com a cópia no servidor. Muitos provedores de git como Github e Gitlab possuem outras funcionalidades, que passam validações ou outras operações em cima do código antes que ele entre em produção, sendo uma funcionalidade essencial para pipelines de integração contínua / deployment contínuo (CI/CD) [5].

O "CI"em CI/CD sempre se refere à integração contínua, que é um processo de automação para os desenvolvedores. CI de sucesso significa que novas mudanças de código em um aplicativo são regularmente construídas, testadas e implementadas em um repositório compartilhado (Normalmente git). É uma solução para o problema de existirem diversos ramos (branches) de um aplicativo em desenvolvimento ao mesmo tempo que podem conflitar, modificando os mesmos arquivos de maneiras distintas [6].

A distribuição contínua (CD) é o ato de implementar mudanças de um desenvolvedor do repositório para a produção de maneira automática. Ele resolve o problema de processos manuais precisarem ser executados para que o código entre em produção, retardando a entrega de um novo patch. Validações contra bugs podem também ser executadas como parte do CI/CD, adicionando mais uma camada de segurança para a aplicação [6]. Para termos uma aplicação com as características de qualidade citadas anteriormente, abrem se uma gama de possibilidades para ser arquitetado o sistema. Desde on premises com a execução em um monolito, ou até uma aplicação completamente serverless rodando containers em um ambiente de computação em nuvem. Cada caso de uso deve ser avaliado para que uma solução compatível seja escolhida.

Containerização é o ato de empacotar código de software com apenas as bibliotecas do sistema operacional (SO) e as mínimas dependências necessárias para executar o código [7]. Com esse recurso, pode ser criado um único executável leve - chamado de container - que roda consistentemente em qualquer infra-estrutura. Atualmente, existem plataformas como o Docker, que ajudam na criação e monitoramento de containers, possibilitando uma implementação rápida e eficiente.

Containers são melhores e mais eficientes em quase todos os aspectos quando comparados a máquinas virtuais (VMs). Em aplicações modernas nativas em ambientes de nuvem, os contêineres se tornaram as unidades de computação padrão [7]. Esse conceito foi explorado para que a aplicação responsável pelo scraping rode de maneira consistente e com autorestauração no provedor de cloud escolhido.

Há várias vantagens em hospedar aplicações em um ambiente de cloud serverless. Uma delas é a característica de ser possível a cobrança apenas do que foi realmente utilizado de computação, dando uma grande flexibilidade para quem arquiteta o sistema. Desde o lançamento das lambda functions na AWS em 2014, o mercado criou uma forte tendência a ter aplicações rodando apenas em ambientes serverless. Lambda functions possibilitaram que pequenos trechos de código sejam rodados de maneira independente de um servidor, sem manutenção ou outras preocupações para o desenvolvedor [1].

Um dos principais recursos que pode hospedado no ambiente de cloud é o Data warehouse. Esse banco de dados se compromete de armazenar dados já processados, disponibilizando as informações para comporem relatórios, dashboards, análises e modelos [1].

Infra-estrutura como Código (IaC) é o gerenciamento de infra-estrutura (Instâncias provisionadas de máquina, redes, bancos de dados, balanceadores de carga, conexões, gateways, etc) em um modelo descritivo, possibilitando toda a infraestrutura de ter versionamento e melhorando o controle sobre todos os recursos e seus gastos. De maneira análoga a um código fonte sempre gerar o mesmo binário, um modelo de infraestrutura como código gera sempre o mesmo ambiente toda vez que é aplicado, sendo algumas linguagens até agnósticas em relação ao provedor de cloud (Google, AWS, Heroku, etc). Infraestrutura como código é uma prática chave de times de DevOps e é muito frequentemente usado em repositórios com entrega contínua, aplicando de maneira automática as novas mudanças especificando adição ou remoção de recursos [8].

Pipelines de dados podem ter diversas fontes, e uma das fontes mais comuns são os próprios websites. Dados disponíveis tanto na própria página, quanto dados disponibilizados para a página por meio de APIs podem ser ingeridos de maneira sistêmica. Nesse tipo de ingestão existem algumas particularidades: Pelo fato do site evoluir e criar novas abas, funcionalidades e layouts, essas aplicações se apresentam mais vulneráveis a falhas e bugs. É essencial que um pipeline de ingestão de dados de websites seja monitorado ativamente, para que tais bugs sejam corrigidos e a aplicação se adapte ao novo estado do site a ser percorrido pelo algoritmo.

2 Materiais e Métodos

Pela sua popularidade dentro do cenário de CS:GO, o site hltv.org foi escolhido como a fonte dos arquivos .dem, tendo em vista que é um dos sites de referência quando se pensa em acessar estatísticas, scoreboards, escalações e etc. A partir das demos, as seguintes tabelas foram escolhidas para serem continuamente atualizadas:

Tabela 1: Conjuntos de dados criados

Tabela	Descrição
Rounds	Dados da partida round a round, com a condição de vitória do round, scores, quantidade de recursos que cada time usou, etc.
Players	Informações de cada jogador ao longo de cada round, como nível de vida, armadura, valor de seus equipamentos, se está ou não está numa zona de objetivo do mapa, etc
Snapshots	Tabela com informações de frames da partida. Vários instantes ao longo de cada round (a cada 4 segundos) têm estatísticas gerais registradas, como: bomba plantada (booleano), numero de jogadores vivos de cada time, quantidade de recursos atuais do time.

2.1 Pipeline de dados orquestrado

Um grafo acíclico direcionado (DAG) foi estabelecido no Airflow, com todas as tarefas necessárias para popular tabelas do data warehouse com dados provenientes das partidas processadas.

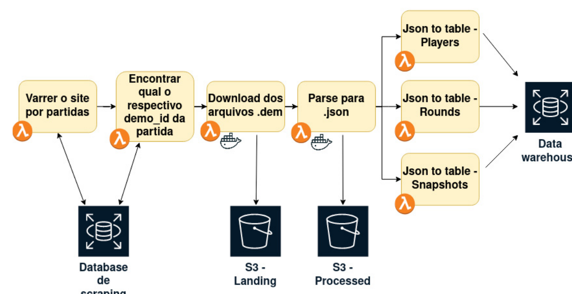

Figura 2: Sequência de tarefas de scraping e tratamento de dados dentro da DAG.

Inicialmente, o scraper percorre o site em busca de partidas que foram recentemente jogadas. Esses dados são registrados em um banco de dados de scraping, que registra cada partida que foi encontrada. Esse banco é necessário para que sejam acessadas apenas as partidas que ainda não tenham sido baixadas.

Em sequência, é iniciada a próxima tarefa: A partir de cada identificador (id) de partida, é possível encontrar o id correspondente à sua demo, acessando uma API do próprio hltv.

Com todos os identificadores das demos já rastreados, é possível iniciar a terceira etapa: baixar os arquivos

.dem e os salvar em um sistema de pastas dentro do Amazon S3 para dados brutos, chamado no projeto de landing zone. É importante destacar a necessidade de um sistema de pastas para armazenamento intermediário, dado que não são dados estruturados que podem ser armazenados em um banco de dados. Ter a possibilidade de reprocessar arquivos já salvos é algo positivo do ponto de vista de manutenção e continuidade do projeto.

Os arquivos estão estruturados dentro do Amazon S3 da seguinte maneira:

s3://jazz-landing/hltv/yyyy-mm-dd/demoname.dem

Essa estrutura permite que diferentes websites sejam explorados, e que cada data em que dados foram coletados possa ser acessada individualmente. A preocupação em separar cada arquivo por sua data é essencial para que custos de varredura de dados sejam menores. Com os arquivos já carregados na landing zone, existe ainda a necessidade de processá-los. A próxima tarefa é transformar arquivos .dem em .json, para que possam ser posteriormente trabalhados em tabelas.

Cada arquivo é processado e retorna para o Amazon S3, desta vez em uma zona chamada de processada. Nesta zona, o arquivo se encontra em um estado intermediário no que diz respeito a tratamento de dados.

Os arquivos na zona processada estão particionados de forma similar, conforme a seguinte estrutura:

s3://jazz-processed/hltv/yyyy-mm-dd/demoname.json

Após essa etapa, os dados já pré processados estão prontos para serem tratados e popular as tabelas do data warehouse. Cada tabela a ser populada deve possuir sua própria task, que carrega para memória em tempo de execução os dados de arquivos .json, os trata e insere no banco de dados PostgreSQL. Vale ressaltar que bancos PostgreSQL não são um padrão em data warehousing, mas foram utilizados no projeto por sua versatilidade.

Vale ressaltar que os arquivos .json gerados nessa etapa possuem uma estrutura complexa, que incorpora informações de diversos aspectos da partida:

- **Configurações gerais da partida**: Mapa, tempos de round, modalidade de jogo
- **Fases da partida**: Começo e inicio de aquecimento, começo e inicio de cada round, halftime, pausas e etc
- **Rounds**: Resultado de cada round, scores, recursos gastos, condição de vitória do time vencedor
- **Damages**: Cada interação de dano causado por um jogador a outro. Posição dos jogadores no momento, arma utilizada, região do corpo atingida, distância relativa entre os jogadores, se o jogador estava afetado por algum efeito de controle de grupo, etc
- **Kills**: Cada interação de neutralização de um jogador por outro. Muito similar às estatisticas de Damages.
- **Weapon fires**: Cada evento de disparo durante a partida
- **Frames**: Momento a momento da partida, posição, velocidade, posição da mira, vida, armadura, entre outras estatísticas relevantes de cada jogador.

A partir desses dados ricos mas ainda brutos, é possível extrair apenas a informação de interesse para cada tabela específica seja criada, de modo a facilitar consultas posteriores. Essas tabelas estão então prontas para serem consultadas via queries e alimentar modelos, relatórios e outros produtos de dados.

Abaixo, um exemplo de consulta na tabela **snapshots** é apresentado, criado a partir dos dados de frames de cada partida.

```
SELECT  "matchID",
        "roundNum",
        tick,
        seconds,
        "clockTime",
        "bombPlanted",
        bombsite,
        "alivePlayers_t",
        "totalUtility_t",
        "alivePlayers_ct",
        "totalUtility_ct"
    FROM  match_data.snapshots
    WHERE created_at >= '2022-05-01'
```

2.2 Implementação em Ambiente de cloud e infraestrutura

Já o provedor de cloud escolhido foi a Amazon Web Services (AWS), por fornecer diversas ferramentas de computação com flexibilidade, e ser um padrão utilizado em grandes empresas de tecnologia.

As ferramentas dentro da AWS utilizadas foram:

- AWS CloudFormation - Gerenciador de infraestrutura como código
- AWS Simple Storage Service (S3) - Sistema de arquivos gerenciado em nuvem
- AWS Elastic Container Service (ECS) - Execução de containers em nuvem
- AWS Elastic Compute Cloud (EC2) - Instâncias de VM em nuvem
- AWS Lambda - Serviço de computação serverless

- AWS Elastic Container Registry (ECR) - Repositório de imagens Docker
- AWS Relational Database Service (RDS) - Banco de dados provisionado em nuvem

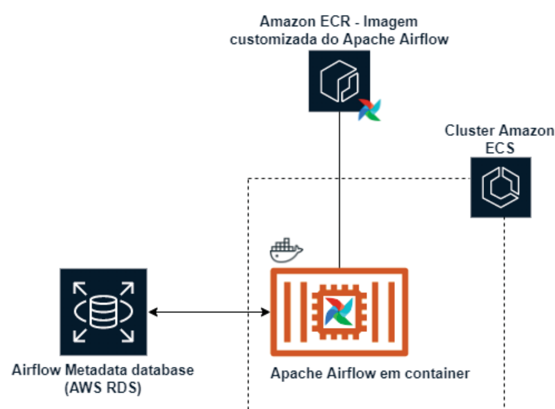

Figura 3: Esquemático mostrando como o Airflow está hospedado.

Para que cada atividade do pipeline seja executada de maneira sequencial, o conceito de DAG foi utilizado dentro do software Airflow, um orquestrador de tarefas muito utilizado no contexto de engenharia de dados. O orquestrador Apache Airflow coordena tarefas unitárias, distribuidas em funções serverless AWS Lambda. Cada etapa da ingestão é monitorável através do Apache Airflow e de logs de execução do AWS Lambda. O Airflow é executado dentro de um container Docker, que permite que o serviço AWS ECS o execute e monitore, mantendo alta disponibilidade do serviço.

Ao definir um cluster de máquinas no serviço ECS, é possível criar um serviço baseado em uma definição de tarefa, que pode conter um ou mais containers. Todos os containers dentro da ECS são monitorados, de modo a serem reiniciados caso necessário para que a aplicação tenha alta disponibilidade.

O Apache Airflow possui três principais partes: Scheduler, webserver e banco de metadados. O scheduler é o agente que define a sequência de tarefas a serem executadas e se comunica com o banco de metadados para registrar as execuções de cada tarefa dentro de cada DAG. Já o webserver é a parte da aplicação com uma interface para o usuário, onde é possível ver execuções, reiniciar tarefas e outras funcionalidades.

Dentro do cluster, um container monitorado ativamente pela Amazon ECS executa uma imagem Docker com algumas modificações feitas em uma imagem clássica do Apache Airflow, através de uma definição de tarefa. Uma tarefa pode envolver um ou mais containers atuando em conjunto para que uma aplicação funcione. Embora o Apache Airflow possa ser executado com suas diversas partes em multiplos containers (scheduler e webserver), foi optado por apenas um container por simplicidade.

O repositório de imagens docker escolhido foi o Amazon ECR, que possui não só a imagem necessária para o Airflow como também para algumas subtarefas do pipeline, que necessitam de mudanças no ambiente para que executem adequadamente.

2.3 Operacionalização - CI/CD

Adicionar novas tarefas e seus recursos em cloud é uma tarefa complexa e morosa quando feita manualmente, principalmente durante o desenvolvimento do código. Por isso, foram adotadas práticas de CI/CD para auxiliar na implementação do projeto.

Toda a integração contínua e entrega contínua (CI/CD) foram feitos utilizando AWS Cloud Development Kit (CDK), uma biblioteca em python de infraestrutura como código que se comunica de maneira nativa com o gerenciador de infraestrutura. A implementação da integração foi feita via Github Actions.

Através do Github Actions, é possível estipular uma série de tarefas a serem desempenhadas a partir de mudanças no código sendo enviadas ao repositório remoto. O CD funciona da seguinte maneira: A partir do momento que o Github Actions é ativado ao enviar mudanças para o repositório remoto, um script responsável por atualizar toda a estrutura é ativado.

Primeiramente, a infraestrutura é comparada com a atual, e caso mudanças sejam encontradas, novos recursos são criados através do CloudFormation, incorporando à aplicação. Além disso, as imagens Docker precisam ser compiladas a cada mudança feita no código que irá executar dentro delas ou na definição do container pela Dockerfile. Esse processo também ocorre de maneira sistêmica no repositório.

O processo torna automática a adição de novas funcionalidades e recursos, além de propiciar uma manutenção mais fácil, por ter um descritivo de tudo que foi adicionado ao ambiente de cloud. Vale ressaltar que a execução de todos os scripts acontecem dentro da plataforma do Github Actions, que disponibiliza esse serviço de maneira gratuita.

O esquemático a seguir mostra o passo a passo de incorporação de infraestrutura e imagens Docker.

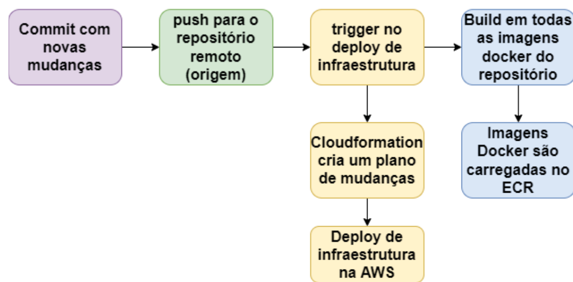

Figura 4: Esquemático mostrando a sequência de operações realizadas pelo script de CI/CD.

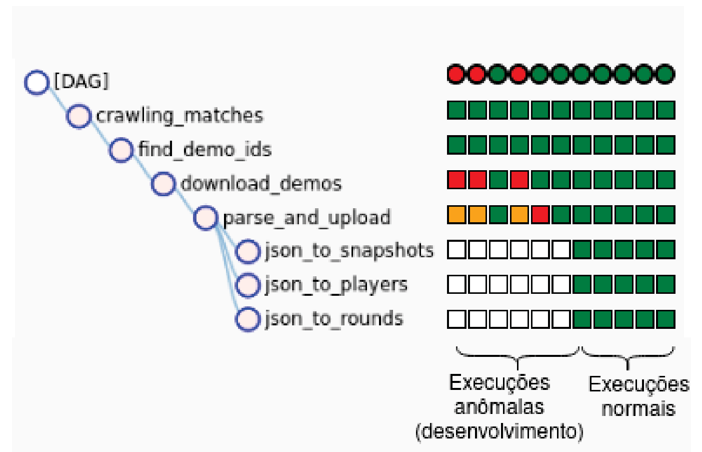

Figura 5: Esquemático mostrando a sequência de tarefas dentro da DAG.

Com todas as imagens atualizadas, os scripts de ingestão e tratamento de dados automaticamente irão executar na versão mais atual do código.

3 Resultados e Discussões

A plataforma proposta foi implementada e o pipeline executado entre os dias 11 e 18 do mês de junho, coletando dados de partidas recentes. Todo o processo, desde scraping até o dado ser inserido no data warehouse, levou tempos entre 40 e 55 minutos.

De forma geral, o resultado do trabalho se apresenta não só na plataforma de dados construída, como também na entrega das tabelas finais e a garantia de sua consistência. Dados de até 20 Gigabytes por dia de dados brutos são processados para arquivos .json, que possuem em média 2 Gigabytes por dia.

Todo esse processamento é acompanhado através do Airflow, que auxilia na observabilidade de quais tarefas foram executadas ao longo do tempo. Na imagem a seguir, um intervalo de 11 dias de execuções são mostrados, sendo cada um representado por uma coluna de quadrados. Durante os primeiros 6 dias do intervalo mostrado, percebem-se erros (vermelho), e execuções não iniciadas (laranja/branco), dado que o fluxo ainda estava sendo desenvolvido. Já na segunda fase, com as tabelas já definidas e testadas, pode se observar todas as indicando sucesso.

Um grande benefício de utilizarmos Airflow é a observabilidade de erros em qualquer parte do pipeline de dados. Caso o site mude sua estrutura, ficaria claro em qual etapa do processo o erro ocorreu, facilitando a manutenção do código e agilizando o retorno das operações ao normal.

Além disso, todos os logs de execução são persistidos na AWS, o que facilita todo tipo de análise em cima de erros e propicia uma maior agilidade ao tratar problemas nas tarefas.

3.1 Proposta de utilização dos dados

A partir dos dados tratados, é possível fazer uma série de análises de modo a entender melhor o comportamento dos jogadores e mapas, bem como outras estatísticas relevantes.

Apesar do foco deste trabalho se tratar da engenharia para trazer e armazenar os dados, foi realizada uma exploração básica das tabelas trazidas, para exemplificar o que um analista de dados poderia criar.

Consequentemente, é possível criar visões e consultas para extrair indicadores, como exemplificado na seguinte consulta:

```
WITH query_base
AS (SELECT
  "clockTime",
  CAST(LEFT("clockTime", 2) AS int)*60
  +
  CAST(RIGHT("clockTime", 2) AS int) AS clock_time,
  "alivePlayers_t",
  "alivePlayers_ct"
FROM match_data.snapshots)
SELECT
  (clock_time - 115)*-1 as round_time,
  AVG("alivePlayers_t"),
  AVG("alivePlayers_ct")
FROM query_base
GROUP BY clock_time
```

Na consulta, é feito o cálculo da média de jogadores vivos a cada segundo da partida, tratando o campo **clockTime** é um campo de texto, com o tempo remanescente do round no relógio, e portanto deve ser tratado para o número de segundos, e posteriormente para uma referência crescente, para que o plot do dado seja mais intuitivo. De maneira direta a partir da consulta acima, é possível plotar o gráfico relacionando tempo de partida e quantidade de jogadores vivos ao longo de cada round.

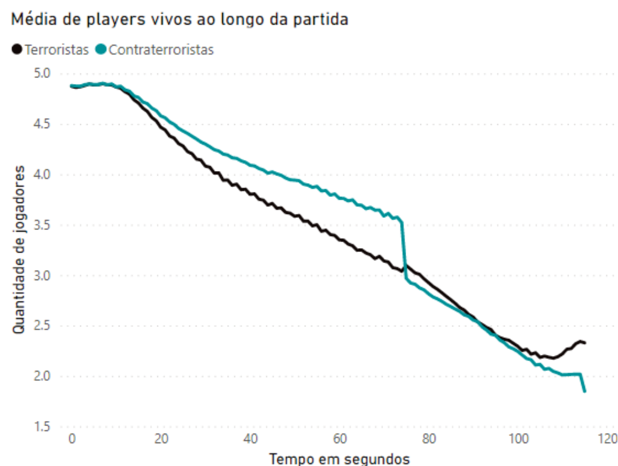

Figura 6: Média da quantidade de players vivos ao longo de cada round.

Neste outro gráfico, a análise é feita em cima de cada um dos mapas e qual dos times é mais favorecido naquele mapa, dado a assimetria de objetivos e espacial de cada mapa. Futuras análises podem se aprofundar nos motivos pelos quais um time seria favorecido em um determinado mapa.

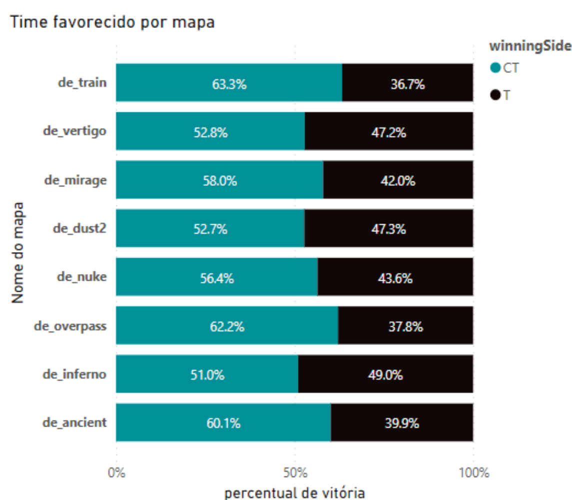

Figura 7: Percentual de vitória de cada um dos times para uma gama de mapas.

Por fim, um histograma de quantidade de rounds dentro de cada partida foi criado. Como o fator que determina o resultado final do jogo é a quantidade total de rounds de um time ultrapassar um determinado número, a quantidade total de rounds da partida pode variar.

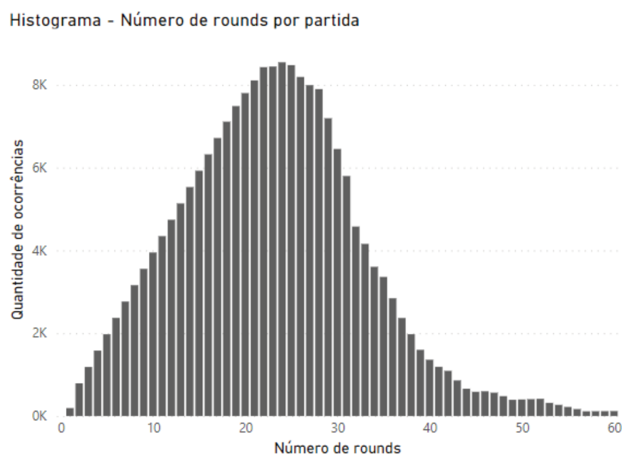

Figura 8: Histograma de quantidade total de rounds jogados a cada partida.

Analistas, cientistas e outros membros envolvidos na análise desses dados podem construir seus objetos de estudo e produtos de dados em cima do banco relacional PostgreSQL, que é atualizado de maneira automática diariamente para que dados frescos de partidas recém jogadas possam incrementar a base existente de dados.

4 Conclusões

Pela perspectiva prática do trabalho apresentado, deve ser destacado que uma fundação sólida em conceitos teóricos é essencial para que um bom produto de dados seja criado. A observabilidade das execuções através do Apache Airflow, a consistência e alta disponibilidade do ambiente em cloud e a agilidade de uma implementação contínua são o núcleo deste trabalho.

Vale ressaltar que muitas outros dados poderiam ser tratados, tabelas correlacionadas e até um modelo de fatos e dimensões criado para analisar por outras diversas perspectivas. Toda essa modelagem dos dados é outro aspecto de engenharia que pode ser abordado em trabalhos futuros, e é extremamente importante para que um relatório possua uma consistência e manutenibilidade.

Agradecimentos

Aos meus amigos e familiares por todo o apoio e carinho, que muito me ajudaram a ter a tranquilidade necessária para realizar este trabalho. Agradeço também aos professores pelas suas correções e ensinamentos, me permitindo aprofundar conceitos e o meu entendimento do tema ao longo do curso. Por fim, agradeço ao Pedro

Augusto, por ter contribuido com dedicação e amizade na condução desse projeto.

Referências

[1] Joe Reis and Matt Housley. *Fundamentals of Data Engineering: Plan and build robust data systems*. O'Reilly Media, Inc., 2022.

[2] Martin Kleppmann. *Designing Data-Intensive Applications: The big ideas behind reliable, scalable and maintainable systems*. O'Reilly Media, Inc., 2017.

[3] Ryan Mitchell. *Web Scraping with Python*. O'Reilly Media, Inc., 2018.

[4] Microsoft Azure. Choose a data pipeline orchestration technology in azure. https://docs.microsoft.com/en-us/azure/architecture/data-guide/technology-choices/pipeline-orchestration-data-movement, 2022. Accessed: 2022-05-27.

[5] Microsoft. What is git? https://docs.microsoft.com/en-us/devops/develop/git/what-is-git, 2022. Accessed: 2022-05-26.

[6] RedHat. What is ci/cd? https://www.redhat.com/en/topics/devops/what-is-ci-cd, 2022. Accessed: 2022-05-26.

[7] IBM Cloud Education. Containerization. https://www.ibm.com/cloud/learn/containerization/, 2022. Accessed: 2022-05-26.

[8] Microsoft. What is infrastructure as code? https://docs.microsoft.com/en-us/devops/deliver/what-is-infrastructure-as-code, 2022. Accessed: 2022-05-23.