UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
ENG. DE CONTROLE E AUTOMAÇÃO

YAN CESCON HAEFFNER - 00273182

ARQUITETURA EM NUVEM ESCALONÁVEL PARA COLETA E PROCESSAMENTO DE DADOS

Porto Alegre
2022

YAN CESCON HAEFFNER - 00273182

ARQUITETURA EM NUVEM ESCALONÁVEL PARA COLETA E PROCESSAMENTO DE DADOS

Trabalho de Conclusão de Curso (TCC-CCA) apresentado à COMGRAD-CCA da Universidade Federal do Rio Grande do Sul como parte dos requisitos para a obtenção do título de *Bacharel em Eng. de Controle e Automação* .

ORIENTADOR:

Prof. Dr. Marcelo Götz

Porto Alegre
2022

YAN CESCON HAEFFNER - 00273182

ARQUITETURA EM NUVEM ESCALONÁVEL PARA COLETA E PROCESSAMENTO DE DADOS

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção dos créditos da Disciplina de TCC do curso *Eng. de Controle e Automação* e aprovado em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____
Prof. Dr. Marcelo Götz, UFRGS
Doutor pela Universität Paderborn - Paderborn, Alemanha

Banca Examinadora:

Prof. Dr. Marcelo Götz, UFRGS
Doutor pela Universität Paderborn - Paderborn, Alemanha

Prof. Dr. Pedro Rafael Bolognese Fernandes, UFRGS
Doutor pela Universität Dortmund - Dortmund, Alemanha

Prof. Dr. João Cesar Netto, UFRGS
Doutor pela Université Catholique de Louvain - Louvain-la-Neuve, Bélgica

_____
Prof. Dr. Mário Roland Sobczyk Sobrinho
Coordenador de Curso
Eng. de Controle e Automação

Porto Alegre, outubro de 2022.

AGRADECIMENTOS

Primeiramente aos meus pais, Rudimar e Marlova, e minha irmã, Yasmine, pelos inúmeros sacrifícios em prol da minha formação, sem os quais jamais teria sido possível aproveitar as oportunidades que tive e chegar até aqui. Agradeço a constante presença, física ou não, e o apoio imensurável que me proporcionaram ao longo de todos esses anos, mesmo de longe.

Agradeço minha namorada, Bruna, pela força, compreensão e confiança nas incontáveis vezes que me fiz ausente pelas obrigações e desafios que tive, por estar comigo desde o início deste desafio e por continuar aqui a assistir o fim de mais um capítulo.

Também agradeço ao meu orientador, Marcelo Götz, por todo o suporte proporcionado e conhecimento compartilhado, assim como pelo interesse e disposição para com o assunto proposto.

Ao Professor Diego Eckhard, que me permitiu ter os primeiros contatos com os desafios na área de dados que hoje tenho imenso apreço, e aos amigos e colegas de trabalho, Paulo Aranha e Tomaz Lago, por terem me ajudado a seguir meu caminho na área.

Ao amigo, colega de trabalho e de faculdade, Alex Treviso, com quem compartilhei percalços, conquistas, dias e noites de estudo ao longo dos anos na faculdade, por todas as revisões e colaborações com este trabalho.

Agradeço também aos amigos Bruno Rolim e André Monteiro que, mesmo sem o contato diário dos primeiros anos de faculdade, nunca pouparam esforços com companheirismo e apoio ao longo destes anos.

Gostaria de agradecer a todos os amigos, colegas de faculdade, de trabalho e familiares que ao longo destes desafiadores anos de faculdade me auxiliaram e ajudaram a fazer desta etapa a experiência mais significativa e marcante de minha vida até aqui.

Sem menor valor, agradeço a Deus pelos caminhos que pude seguir, pelas amizades que fiz, pelos fracassos e pelas lições que com eles aprendi.

“A experiência é um troféu composto por todas as armas que nos feriram."
Marco Aurélio

RESUMO

Este trabalho tem como objetivo o desenvolvimento de uma arquitetura capaz de processar dados de diferentes sistemas de forma autônoma, eficaz e escalonável, permitindo, assim, a inclusão de novas unidades destes sistemas, sem a necessidade de alteração da arquitetura e de seus componentes. Por meio dos serviços da Amazon Web Services e do orquestrador de tarefas agendadas Apache Airflow foi possível automatizar o paralelismo da execução dos processos (desenvolvidos em Python para utilização no Airflow) de extração, ingestão, transformação e agregação de dados de estações meteorológicas disponibilizadas pelo Instituto Nacional de Meteorologia e de estações de irrigação automática desenvolvidas em um trabalho de conclusão de curso anterior a este. No desenvolvimento, todos os processos foram orquestrados de maneira que os dados fossem centralizados em um banco relacional analítico para consumo imediato e especializado. O acesso às estações disponíveis contou com o uso dos protocolos HTTP e MQTT, que permitiu a obtenção de dados em diferentes frequências de execução para estações localizadas em diferentes cidades e estados do país. Visando ao enfoque dos benefícios da engenharia de dados para projetos e demandas que necessitam de informações de diferentes sistemas, avaliou-se a configuração dos controladores projetados para as estações de irrigação mediante verificação do comportamento daquelas frente aos dados meteorológicos mais próximos disponíveis. Por fim, a arquitetura em análise se mostrou bastante eficaz no escalonamento de unidades de processamento para a execução das diversas tarefas necessárias utilizando serviços de conteinerização de código (como o Amazon Fargate) e armazenamento distribuído em nuvem (como o Amazon S3) sem prejudicar ou alterar os dados trabalhados ao longo dos processos utilizados.

Palavras-chave: Computação em nuvem, IoT, Engenharia de Dados, Amazon Web Services, Integração de Sistemas.

ABSTRACT

This work aimed to propose a cloud architecture capable of processing data from different systems in an autonomous, efficient and scalable way, thus allowing the addition of new units of those systems without interfering with the architecture and its components. By making use of different services of Amazon Web Services and the orchestrator of scheduled tasks, Apache Airflow, it was possible to automate the parallelism of the execution of different processes, developed in Python, of extraction, ingestion, transformation and aggregation of data from meteorological stations publicly available by the National Institute of Meteorology API and automatic irrigation stations developed in an undergraduate thesis prior to this one. All processes were orchestrated so that the data was centralized in an analytical relational database for on-demand and specialized consumption. The access to the available stations used the HTTP and MQTT protocols, allowing data to be extracted at different frequencies of execution for stations located in different cities and states of the country. Still, seeking to bring light to the benefits of data engineering in projects and demands with different systems, it was possible to evaluate the configuration of the controllers designed for the irrigation stations by verifying their behavior against the closest available meteorological data. Finally, the proposed architecture proved to be effective in scaling processing units to perform the various necessary tasks using code containerization services, such as Amazon Fargate, and distributed cloud storage such as Amazon S3, while keeping the data integrity through all the necessary processes.

Keywords: IoT, Cloud, Data Engineering, Apache Airflow, Amazon Web Services, ELT, Automation.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

CCA	Curso de Eng. em Controle e Automação
IoT	Internet of Things
M2M	Machine-to-Machine
INMET	Instituto Nacional de Meteorologia
HTTP	Hypertext Transfer Protocol
MQTT	Message Queuing Telemetry Transport
TCP	Transmission Control Protocol
IP	Internet Protocol
OSI	Open System Interconnection
API	Application Programming Interface
AWS	Amazon Web Services
EC2	Elastic Compute Cloud
SaaS	*Software as a Service*
IaaS	*Infrastructure as a Service*
PaaS	*Platform as a Service*
S3	Simple Storage Service
SQS	Simple Queue Service
ECS	Elastic Container Service
ECR	Elastic Container Registry
DAG	*Directed Acyclic Graph*
JSON	*JavaScript Object Notation*
SQL	*Standard Query Language*
ETL	*Extract, Transform and Load*
ELT	*Extract, Load and Transform*

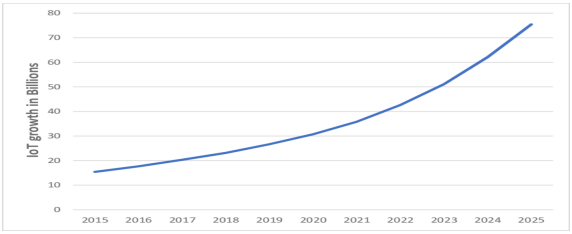
LISTA DE SÍMBOLOS

hPa	hectopascal
^{o}C	graus Celsius
º	graus de latitude e longitude
kJ/m^2	quilojoules por metro quadrado
mm	milímetro
m/s	metros por segundo
V	Volts
$hav(\Theta)$	metade do seno verso de um ângulo
$\Delta\psi$	diferença de latitude entre dois pontos
$\Delta\lambda$	diferença de longitude entre dois pontos
ψ	latitude em radianos
λ	longitude em radianos
R	raio médio da terra, em quilômetros
d	distância entre dois pontos em uma esfera, em metros

1 INTRODUÇÃO

Os avanços tecnológicos dos processos de automação e comunicação M2M (do inglês *Machine-to-Machine*) trouxeram consigo tanto novas soluções quanto novos desafios para seus mais diversos setores de aplicação. A Figura 1 mostra como o número de dispositivos inteligentes vem crescendo de maneira notável no mercado, de forma que se torna impossível ignorar os motivos e as consequências do uso dos mesmos nas mais diversas soluções da humanidade.

Figura 1: *Número de dispositivos IoT em bilhões em operação por ano (extrapolados).*



Fonte: (BALI et al., 2021)

Inicialmente vistos apenas como subproduto destes avanços tecnológicos, os dados gerados por estes processos trouxeram consigo a possibilidade de uma nova visão sobre a maneira de pensar e desenvolver soluções (MILLER; MORK, 2013). Na agricultura, por exemplo, um dos desafios que marca a revolução tecnológica do setor é a necessidade de conciliar os dados gerados pelo uso de dispositivos inteligentes com a inteligência do negócio em si (HIMESH, 2018).

Uma das grandes vantagens, reconhecida nos últimos anos, do uso de dados para a geração de valor de mercado se dá na possibilidade de agregar e correlacionar informações de naturezas distintas para objetivos e soluções específicas (GÜNTHER et al., 2017). Contudo, dados isolados não são capazes de fornecer informações suficientes para esclarecer e responder as complexas perguntas que compõe o ambiente no qual a solução está inserida, sendo necessário processar e agregar estas informações para colher o valor desejado (SAGGI; JAIN, 2018).

Por consequência, a crescente busca pela utilização de dispositivos e sistemas que são capazes de gerar dados também tem como contraponto a crescente complexidade dos processos de captura e processamento destas informações (KHAN et al., 2014). A busca por soluções capazes de se adequarem a diferentes volumes operacionais trava batalhas em diversas frentes tecnológicas, sendo o uso de serviços e sistemas distribuídos na nuvem um

dos grandes destaques do setor dada sua capacidade de reduzir custos e otimizar processos para os desafios mencionados (YANG et al., 2016).

Este trabalho é motivado pela necessidade de apresentar e testar possíveis soluções práticas utilizando tecnologias atuais para os problemas de integração e processamento descritos anteriormente, os quais ganham cada vez mais visibilidade dada a crescente adoção à tomada de decisões baseadas em dados dos mais diversos setores econômicos.

Como parte de um estudo de caso, este trabalho fará uso de sistemas autônomos de irrigação desenvolvidos em um trabalho de conclusão de curso de Engenharia de Controle e Automação anterior à este. A utilização de tal sistema se dá pela necessidade de integração e processamento das informações geradas para agregar valor de inteligência operacional à solução, reforçando assim a continuidade dos processos de automação.

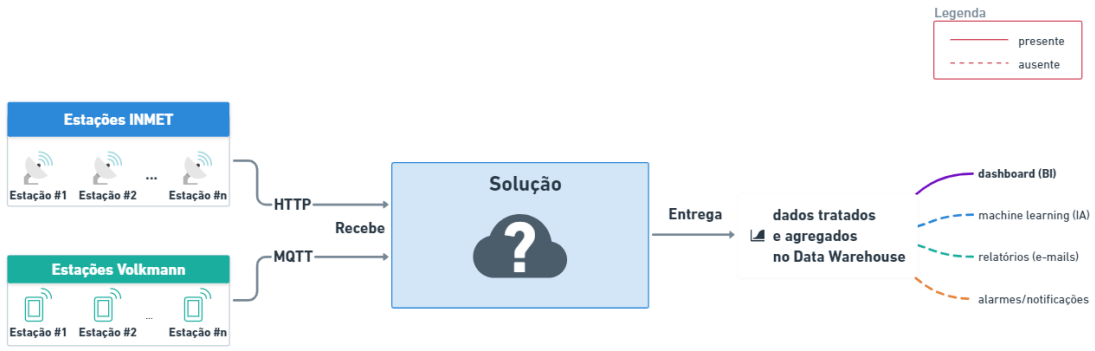
1.1 Objetivos

Buscando validar uma solução que permita a integração de sistemas distintos, propõe-se uma arquitetura de processamento de dados na nuvem que seja facilmente adaptável a futuras implementações, problema muito comum em um mundo que está começando a caminhar em direção a decisões baseadas em dados. Ainda, as tarefas devem ser autônomas, observáveis e de fácil monitoramento, reduzindo a intervenção humana.

Para demonstração e prova de conceito, serão processados dados provenientes de estações climáticas públicas, fornecidos pelo INMET, e sistemas autônomos de irrigação, controlados por microcontroladores, já em operação. Para a obtenção destes dados serão desenvolvidas estruturas de código escritas na linguagem de programação *Python*, utilizando protocolos HTTP (do inglês *Hypertext Transfer Protocol*) e MQTT (do inglês *Message Queuing Telemetry Transport*) em conjunto com soluções em nuvem fornecidas pela AWS (Amazon Web Services) e *scripts* SQL para desenvolvimento de estruturas em um banco de dados relacional.

A Figura 2 apresenta a estrutura geral do problema ao qual a arquitetura deste trabalho pretende resolver.

Figura 2: *Escopo do problema abordado neste trabalho.*



Fonte: O autor.

Ainda, como forma de ressaltar o valor inerente do processo de agregação de dados, serão cruzados os dados das estações de irrigação com os dados das estações meteorológicas mais próximas para avaliar a correta instalação do sistema e configuração dos controladores, ambas etapas já desenvolvidas em um trabalho anterior à este.

2 REVISÃO BIBLIOGRÁFICA

Nesta seção, serão descritos os principais conceitos relacionados tanto às tecnologias quanto às metodologias adotadas ao longo do desenvolvimento do trabalho.

2.1 Internet of Things

O termo "Internet das Coisas"(do inglês *Internet of Things* - IoT) se trata de um paradigma em que dispositivos são capazes de transportar e coletar informações via internet, muito comumente por meio dos protocolos TCP/IP (do inglês *Transmission Control Protocol/Internet Protocol* (RAYES; SALAM, 2017). A partir dessas informações é possível permitir intercomunicação de dispositivos, tomada de decisões, acionamentos, previsões e monitoramento de sistemas remotos com praticidade e agilidade.

Apesar do nome IoT ter sido introduzido pela primeira vez ao mundo em 1999, o conceito de comunicação de dispositivos através da comunicação remota já permeava as necessidades da indústria e de diferentes negócios desde 1970, tendo a primeira comunicação efetiva de um dispositivo com a internet ocorrido em 1990 (ROSE; ELDRIDGE; CHAPIN, 2015). Essa capacidade de comunicação marcou um ponto histórico para a humanidade, visto que revolucionou a maneira de vida de todos deixando casas, meios de transporte, indústrias e cidades cada vez mais inteligentes e interligadas.

2.2 Comunicação por MQTT

MQTT (do inglês *Message Queue Telemetry Transport*) é o nome do protocolo que atua na camada de aplicação de um modelo OSI de transferência de dados sobre o protocolo TCP/IP (EUROTECH, 1999). Dada sua operabilidade assíncrona, em que não é necessário que o receptor da mensagem esteja sincronizado com o emissor, tornou-se um dos meios de comunicação de dados mais utilizados em estruturas IoT, pois permite o funcionamento destas estruturas mesmo em conexões não confiáveis (QUINCOZES; EMILIO; KAZIENKO, 2019).

O funcionamento assíncrono deste protocolo se dá graças ao uso de uma fila de mensagens, endereçadas por tópicos, que é consumida pelos receptores de acordo com o tópico ao qual estão cadastrados. O servidor que hospeda as estruturas necessárias para o funcionamento desta fila é denominado *broker*, enquanto os elementos que leem e escrevem nessa fila são denominados clientes (YACCHIREMA; PALAU; ESTEVE, 2017).

Esse tipo de comunicação é muito utilizado em soluções IoT que demandam troca de informações de forma *online*, já que permite que dispositivos alimentem a fila de mensagens a qual estão associados sem interromper o funcionamento daqueles (QUINCOZES; EMILIO;

KAZIENKO, 2019).

2.3 Comunicação por HTTP

Assim como o MQTT, o protocolo HTTP (do inglês *Hypertext Transfer Protocol*) também atua na camada de aplicação de um modelo OSI, sobre o protocolo TCP/IP (GRIGORIK, 2013). Tem como principal característica a comunicação cliente-servidor, onde o cliente primeiro faz a requisição (nomeada *request*) para então obter a resposta (nomeada *response*) do servidor com a informação desejada.

A comunicação entre cliente e servidor é feita através de requisições de tipo definido que, através do auxílio de cabeçalhos, permitem a interpretação da mensagem de maneira inteligente tanto pelo servidor quanto pelo cliente (GRIGORIK, 2013). Exemplos importantes dos tipos de requisição são:

- GET: métodos que indicam o consumo de determinada informação do servidor.
- POST: métodos que indicam o envio de determinada informação ao servidor.

Apesar de ter surgido em 1990, este protocolo passou por revisões e melhorias ao longo dos anos, tais como encapsulamento, criptografia dos dados transmitidos e um melhor gerenciamento de sessões, que permitiram sua implementação em um número cada vez maior de soluções, principalmente por meio de aplicações *web* estruturadas (BRYLINSKI; BHATTACHARJYA, 2017).

2.3.1 APIs

Conhecidas como APIs (do inglês *Application Programming Interface*), rotinas comportamentais de código devidamente estruturadas e, normalmente, acessíveis por meio do protocolo HTTP, ganharam importância na integração de sistemas devido à suas características de escalabilidade e praticidade de implementação como serviços modulares e independentes (HASSAN; BAHSOON; BUYYA, 2022).

Estas soluções são serviços de comunicação cliente-servidor que permitem a integração de leitura/escrita entre sistemas de maneira rápida, modular e previsível. O acesso a estas aplicações é garantido através de um endereço *web* e gerenciado por meio de mecanismos de segurança e criptografia inerentes do protocolo HTTP (com a possibilidade de camadas de segurança extras gerenciadas pelo próprio servidor da aplicação) (VUKOVIC, 2015).

Por ser uma solução que permite a integração de um sistema como um serviço, onde o consumidor precisa respeitar regras inerentes do sistema para troca de informação, a utilização de APIs no cenário IoT é cada vez mais frequente para transferência de dados baseadas em eventos.

2.4 Modularização por Contêineres

Com os avanços da computação distribuída, também foram necessárias inovações no gerenciamento do ciclo de vida da execução de uma aplicação em ambientes distribuídos. A prática de empacotar *software*, com todas as dependências necessárias para execução e de maneira isolada em um sistema operacional, é denominada conteinerização, enquanto que as estruturas que armazenam, isolam e executam estes códigos, são chamadas de contêineres.

A utilização deste tipo de tecnologia em ambientes distribuídos permite que aplicações possam ser executadas de maneira contínua e escalável em um número virtualmente ilimitado de máquinas (AL-RAKHAMI et al., 2019), sem a necessidade de gerenciar pré-configurações de maneira equivalente para que as mesmas condições de operação sejam alcançadas. Ainda, estas aplicações se beneficiam do completo isolamento de outras aplicações sendo gerenciadas pelo mesmo sistema operacional, dando mais segurança e eficiência para o uso compartilhado de recursos da máquina (AL-RAKHAMI et al., 2019).

Graças a estas características, a implementação de soluções e arquiteturas que se beneficiam de elementos contêinerizados vem crescendo em diferentes áreas com benefícios notáveis, desde soluções automotivas (KUGELE; HETTLER; PETER, 2018) até na área da saúde (ANDRY; RIDOLFO; HUFFMAN, 2015).

2.5 Computação em Nuvem

Apesar de o termo “Computação em Nuvem"(do inglês *Cloud Computing*) ser utilizado para descrever o uso de componentes de computação distribuídos por meio da internet desde a década de 90 (SURBIRYALA; RONG, 2019), ainda não existe uma definição universal para este tipo de tecnologia.

Ainda assim, apesar desta indefinição, a utilização de recursos computacionais remotos se tornou muito popular após o lançamento do serviço de aluguel de máquinas virtuais da AWS, o Elastic Compute Cloud (EC2). Em suma, a computação em nuvem permite que empresas paguem por infraestrutura, serviços e tecnologias sem precisar se preocupar com custos de manutenção, desvalorização de equipamentos físicos e eventuais acidentes ou condições naturais adversas. Estas características de computação sob demanda reforçam duas qualidades valiosas para inúmeras soluções e negócios, sendo elas:

- **Escalabilidade**: define a propriedade de expansão da capacidade de processamento de uma solução impactando diretamente os custos desta solução. Na computação em nuvem, a previsibilidade gerada pela ausência dos custos de manutenção e falhas físicas de sistemas permite antecipar os custos em cenários de maior demanda computacional, otimizando a escalabilidade da solução (LEHRIG; EIKERLING; BECKER, 2015). A escalabilidade é dita horizontal quando é possível distribuir o processamento em um número adequado de operadores (*hardware*) para suprir a demanda, enquanto que uma escalabilidade vertical necessita da melhoria da instância atual (não sendo possível compartilhar tarefas com outros operadores) para adequar o poder de processamento à demanda.

- **Elasticidade**: define a capacidade de uma solução de expandir ou reduzir seus recursos conforme a demanda computacional, evitando desperdício destes recursos sem comprometer a eficiência do serviço prestado. Como na computação em nuvem é possível pagar apenas pelo tempo de utilização de um recurso, a elasticidade pode se fazer presente desde a concepção da arquitetura de uma solução, otimizando os custos envolvidos a longo prazo (LEHRIG; EIKERLING; BECKER, 2015).

2.5.1 Amazon Web Services

Dentre as inúmeras soluções de computação em nuvem, alguns provedores de serviços se destacam pela ampla adoção em inúmeras soluções e negócios no mundo inteiro. Um destes provedores é a Amazon Web Services, que fornece diferentes softwares (SaaS),

infraestruturas (IaaS) e plataformas (PaaS) como serviço (do inglês *Software as a Service*, *Infrastructure as a Service* e *Platform as a Service*).

Alguns dos serviços disponibilizados pela empresa são:

- **Armazenamento distribuído**: o ***Amazon Simple Storage Service (S3)*** é o serviço de armazenamento distribuído na nuvem da AWS. Todos arquivos hospedados no S3 são tratados como objetos, os quais possuem propriedades (metadados) associadas que podem ser verificadas antes de acessar tais objetos.

  Os objetos são armazenados dentro de *buckets*, que são estruturas isoladas dentro da nuvem por regiões de acesso. Apesar de um *bucket* ser alocado em uma região geográfica específica, seu nome é global e único, sendo assim, não é possível nomear dois *buckets* igualmente em toda a nuvem AWS, mesmo que estejam em contas ou regiões diferentes. Cada *bucket* também possui políticas de segurança restritas, sendo necessário garantir o acesso necessário aos serviços/usuários desejados para que possam manipular os objetos.

  Apesar de não operar com o conceito de pastas, a separação dos objetos se dá pelo seu prefixo, o qual permite a nomeação semelhante a um caminho de diretórios, caso necessário, permitindo melhor organização e navegação entre objetos.

- **Bancos Relacionais**: o ***Amazon Relational Database Service (RDS)*** é uma coleção de serviços gerenciados de banco de dados da AWS. Permite a utilização de diversos bancos relacionais com ou sem servidor, reduzindo assim os custos de manutenção e possíveis problemas de segurança ou disponibilidade.

- **Computação na nuvem**: O ***Elastic Compute Cloud (EC2)*** é o serviço de computação distribuída na nuvem AWS por meio de aluguel de máquinas virtuais. O usuário pode escolher, dentre uma lista de opções pré definidas, as configurações da máquina que deseja, as configurações de segurança, de rede e de cobrança da mesma, sendo possível pagar sob demanda ou até mesmo reservar a máquina por um período específico de tempo. Para este serviço, a AWS não se responsabiliza por falhas de software na máquina, visto que o gerenciamento da mesma deve ser feito pelo usuário. A AWS apenas garante o funcionamento e estabilidade da infraestrutura que disponibiliza a máquina.

- **Gerenciamento de Contêineres**: o ***Amazon Elastic Container Service (ECS)*** é o serviço de gerenciamento de contêineres do tipo Docker da nuvem AWS. O gerenciamento permite fácil execução, escalabilidade e performance em redes computacionais de máquinas EC2 ou na modalidade *Fargate*, permitindo assim a inicialização e distribuição de aplicações em contêineres.

- **Registro de Contêineres**: o ***Amazon Elastic Container Registry (ECR)*** é o serviço de armazenamento de contêineres do tipo *Docker* na nuvem AWS. O armazenamento é feito de maneira gerenciada e possibilita o versionamento de um mesmo contêiner através de *tags*, que podem ser acessadas conforme a necessidade do usuário. Assim, é possível manter diversas versões de contêineres de uma mesma aplicação que podem ser gerenciadas, executadas e recuperadas conforme necessário através de sistemas na nuvem ou até mesmo locais por meio de permissões adequadas de usuários na nuvem AWS, caso o registro seja privado, ou por qualquer usuário em caso de registro público.

- **Computação em contêiner sem servidor**: o ***Amazon Fargate*** trata-se de um mecanismo de execução de computação por contêineres sem servidor da nuvem AWS. Este mecanismo permite a utilização de *clusters* ECS para prover, executar e encerrar aplicações em contêineres sob demanda, sem a necessidade de pagar pelo tempo ocioso destas máquinas, o que não ocorre no uso do ECS com máquinas EC2. A grande vantagem deste mecanismo é a possibilidade de executar aplicações sob demanda, sem a preocupação de manutenção e gerenciamento da infraestrutura destas máquinas, e sem pagar pelo tempo em que estas não são utilizadas. Esse tipo de comportamento fornece uma grande vantagem à arquiteturas que dependem de poder de processamento pontual e de forma não contínua, visto que facilitam e tornam mais barato o uso de elasticidade na solução (BURKAT et al., 2021).

2.6 Apache Airflow

As inúmeras integrações necessárias em ambientes e soluções modernas originam, por consequência, inúmeras operações que devem ser executadas diariamente, tornando o trabalho manual impraticável.

Para satisfazer a necessidade de automação de tarefas agendadas de maneira eficiente, em 2014 a empresa Airbnb, que fornece serviços de busca e reserva de imóveis para o mundo todo, desenvolveu uma ferramenta especializada na orquestração de distintos fluxos de trabalho, nomeada Airflow (SINGH, 2019). Em 2016, esta ferramenta foi integrada às soluções da organização sem fins lucrativos Apache, passando a ser conhecida por Apache Airflow e tornando-se uma solução de código aberto.

Em suma, trata-se de um conjunto de sistemas que trabalham em conjunto para garantir o agendamento, execução, distribuição e monitoramento de diversos fluxos de trabalho que podem conter inúmeras tarefas, dependentes ou não entre si.

Cada conjunto de tarefas é definido como um *pipeline*, ou DAG (do inglês *Directed Acyclic Graph*), que é executado do início ao fim sem que existam retornos ou laços de repetição internos (daí a definição de gráfos acíclicos direcionados). Tanto as tarefas individuais quanto os DAGs são desenvolvidos utilizando a linguagem de programação *Python* e são interpretados em tempo de execução pelo orquestrador.

É possível definir o tipo de operador desejado para uma tarefa. O Airflow permite a utilização de inúmeros operadores que definirão a natureza da operação, sendo os mais comuns:

- ***DummyOperator***: operador utilizado apenas como marcação, não executa nenhum algoritmo e é comumente utilizado apenas para delimitar passagens específicas da sequência das tarefas em um DAG.

- ***PythonOperator***: operador que indica a execução de uma função *Python*.

- ***BranchPythonOperator***: operador que indica a execução de uma função *Python* cujo retorno será utilizado para bifurcar a execução das tarefas seguintes.

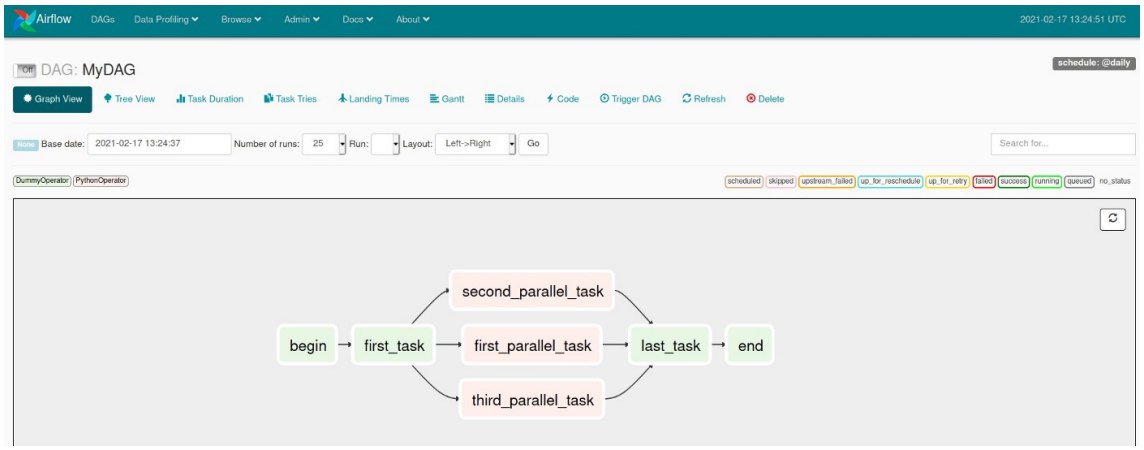
De maneira geral, uma instância funcional do Apache Airflow contém os seguintes componentes:

- **Servidor Web**: componente responsável pela interface gráfica do usuário e apresentação da mesma através de uma página web privada, acessada com as credenciais do usuário desejado.

- ***Scheduler***: responsável pelo controle de tarefas, peça fundamental para o correto funcionamento do orquestrador. Valida o código estrutural das rotinas programadas e verifica o estado atual de cada tarefa agendada, definindo então o próximo estado do sistema a partir das informações atuais.
- **Banco de Dados**: componente responsável por armazenar metadados relacionados tanto às outras estruturas do Airflow quanto das tarefas sendo executadas por ele.
- ***Workers***: unidades que executam de fato as tarefas agendadas, são responsáveis pela execução do código e das rotinas destinadas à tarefa.
- ***Executor***: Componente responsável pelo direcionamento das tarefas solicitadas pelo *scheduler* aos *workers*. Por padrão, utiliza o próprio *scheduler* como *worker*, executando as tarefas sob a mesma máquina, porém, pode ser alterado para diversos outros modelos, permitindo execução isolada das tarefas em contêineres ou até mesmo em máquinas remotas.

A Figura 3 apresenta a visualização de um DAG, através do servidor web, com sua sequência de tarefas definida. Apesar de a ordem de execução ser definida pelo *scheduler*, os nomes das tarefas foram definidos de forma que auxiliem a compreensão do leitor para o fluxo de operações representado graficamente pelo DAG. O processo de exemplo inicia com a tarefa genérica "*begin*", que, ao ser finalizada, aciona a tarefa "*first_task*". Ambas são *DummyOperators* e não executam nenhum algoritmo, apenas servem de marcação visual das etapas. As tarefas "*first_parallel_task*", "*second_parallel_task*" e "*third_parallel_task*" executam algoritmos em *Python* simultaneamente, e, após a execução das três as tarefas de marcação (*DummyOperator*), "*last_task*" e "*end*" são executadas sequencialmente. Ainda, a imagem apresenta o elemento *schedule* no canto superior direito, que representa o agendamento da execução do processo como diário (do inglês *daily*).

Figura 3: *Captura de tela da visualização das tarefas de um DAG genérico.*

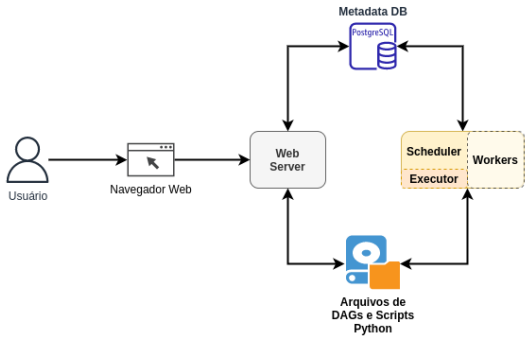


Fonte: O autor.

A Figura 4 apresenta a relação entre os componentes em sua configuração básica, com todos sendo executados em uma mesma máquina local.

É importante ressaltar que esta ferramenta foi idealizada e projetada como um orquestrador de tarefas agendadas, não devendo ser utilizado como processador de dados ou

Figura 4: *Relação entre os componentes do Apache Airflow em configuração padrão.*



Fonte: O autor.

como unidade de processamento completo de tarefas de alto desempenho (onde há compartilhamento dos dados processados em memória). Isso acontece, principalmente, pelo fato de o banco de dados de metadados responsável pela troca de informação entre seus componentes não ser otimizado para o processamento de alta demanda de dados. Ainda assim, é possível fazer uso do banco de metadados para trocar informações relevantes entre tarefas, como datas, resultados, entre outras informações, de operações executadas.

Ainda, como estes componentes não são livres de limites de capacidade de processamento, e como tal limite pode ser facilmente alcançado com integrações complexas e de alta frequência de dispositivos IoT, mesmo quando utilizando o Airflow apenas como orquestrador (LI; ZOU, 2021), há a possibilidade de modularizar seus componentes, exceto o banco de dados de metadados, para a utilização escalável sob demanda através de contêineres.

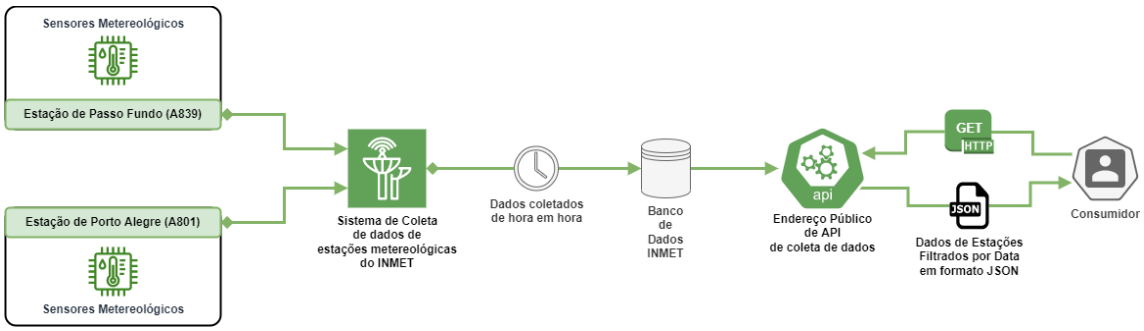
2.7 Estações Meteorológicas

O Instituto Nacional de Meteorologia (INMET) fornece, de forma pública, dados provenientes de suas estações meteorológicas localizadas em diversas cidades do país. Estas estações são elementos físicos equipados de diversos sensores capazes de registrar valores instantâneos de chuva, temperatura, umidade, ponto de orvalho, pressão, vento e radiação.

Através de uma API disponibilizada, o consumidor informa uma data de inicio e fim (dia, mês e ano) para a consulta, assim como um código identificador da estação desejada. O retorno desta API se dá por uma estrutura em formato JSON (do inglês *JavaScript Object Notation*), que possui informações colunares referentes à hora da extração da informação assim como os valores instantâneos dos diversos sensores presentes na estação.

Um fluxograma de dados provenientes de duas estações utilizadas para obtenção de dados, localizadas nas cidades de Passo Fundo e Porto Alegre no estado do Rio Grande do Sul, com seus respectivos códigos de identificação, é apresentado na Figura 5, descrevendo o funcionamento do sistema fornecido pelo INMET para o consumo destes dados.

Figura 5: *Fluxo de dados provenientes das estações meteorológicas do INMET.*



Fonte: O autor.

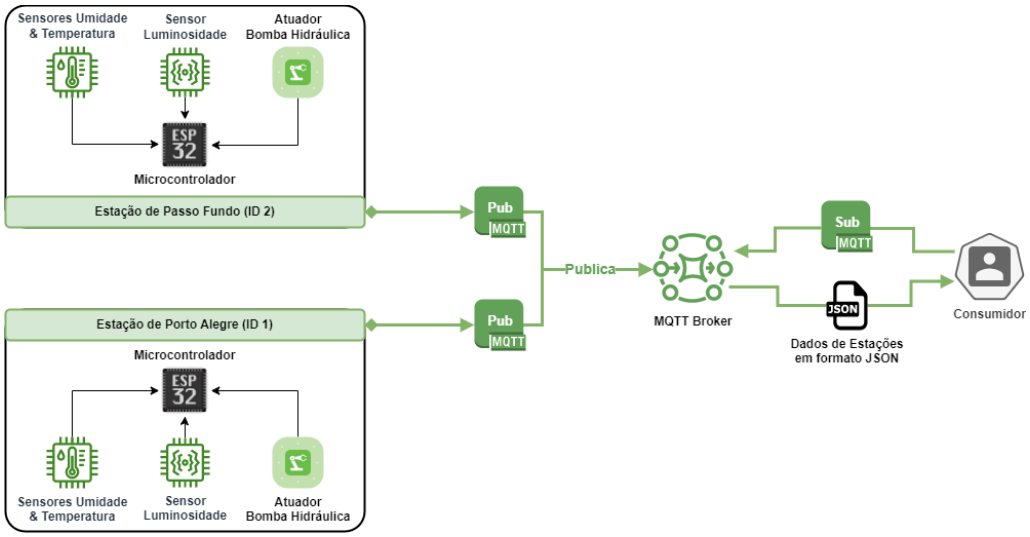
2.8 Estações Automáticas de Irrigação

As estações de irrigação são sistemas embarcados com sensores de luminosidade, temperatura e umidade, assim como atuadores (bombas hidráulicas) que permitem a irrigação automatizada de terrenos abertos de acordo com os níveis de umidade desejados e configurações previamente definidas.

As estações utilizadas foram desenvolvidas como trabalho de conclusão de curso em Engenharia de Controle e Automação pela Universidade Federal do Rio Grande do Sul (VOLKMANN, 2022) utilizando o microprocessador ESP32, que está programado para permitir a publicação de mensagens contendo as leituras dos sensores e eventos dos atuadores através do protocolo MQTT.

A Figura 6 apresenta o fluxograma de dados provenientes destas estações, utilizando um *broker* genérico para representação da comunicação por mensagens.

Figura 6: *Fluxo de dados provenientes das estações de irrigação com Broker genérico.*



Fonte: O autor.

3 MATERIAIS E MÉTODOS

Para permitir a análise de uma arquitetura de ingestão e processamento de dados, se faz necessário desenvolver as estruturas individuais responsáveis por cada uma das etapas existentes até a disponibilização final dos dados agregados. Neste capítulo será apresentado o desenvolvimento das soluções necessárias para extrair, transformar e agregar os dados provenientes de estações meteorológicas públicas e acionamentos e leituras de sensores e atuadores de estações automáticas de irrigação.

O projeto faz uso de duas estações de irrigação e duas estações meteorológicas, localizadas nas cidades de Passo Fundo e Porto Alegre, ambas no estado do Rio Grande do Sul, que fazem o envio de suas informações por meio de estruturas que utilizam os protocolos HTTP e MQTT. Cada estação envia também seu código de identificação, que deverá ser utilizado para o correto isolamento das informações que serão agregadas por região. Este tipo de identificação também permite definir configurações de unicidade de dispositivos durante a configuração e estruturação do banco de dados que armazenará e cruzará os dados desejados.

3.1 Definição do Serviço de Computação em Nuvem

Como o uso de serviços de computação em nuvem é parte essencial para este projeto, será escolhido um provedor dentre os principais disponíveis para o desenvolvimento das etapas que seguem neste trabalho. Apesar de ser possível utilizar diferentes serviços de diferentes provedores em uma mesma arquitetura, esse tipo de abordagem adiciona complexidade tanto na escolha e compatibilidade entre esses serviços quanto nas configurações de segurança necessárias em um ambiente na nuvem.

A Tabela 1 apresenta características de três provedores amplamente utilizados em serviços de nuvem que podem ser admitidos para o desenvolvimento deste trabalho. As informações de cada provedor foram retiradas da documentação publicamente disponível de cada um, enquanto que a comparação de custos foi feita através de calculadoras orçamentárias disponibilizadas por cada provedor, tomando como base os serviços mais comumente utilizados (computação genérica, armazenamento, bancos de dados e redes privadas).

Os serviços pesquisados da plataforma da Google apresentaram os menores custos de utilização, porém o plano gratuito de 12 meses da AWS foi utilizado como ponto de corte para adotar os serviços da AWS no desenvolvimento deste trabalho. A possibilidade de utilizar serviços gratuitos por mais tempo permite manter e comparar arquiteturas conforme necessário para o projeto, sem afetar de forma significativa a solução final.

Sendo assim, todos os serviços em nuvem em uso neste trabalho serão considerados apenas dentro dos limites e condições impostos pela AWS, mas é importante destacar que

Tabela 1: *Comparação entre provedores de computação em nuvem*

Serviço	Amazon Web Services	Google Cloud Provider	Microsoft Azure
Público	Sim	Sim	Sim
Serviços sob demanda	Sim	Sim	Sim
Plano Gratuito	12 meses	$300 nos primeiros 30 dias	$200 nos primeiros 30 dias
Custo	$$	$	$$$

Fonte: O Autor.

existem ferramentas semelhantes ou equivalentes em todas as três plataformas comparadas.

3.2 Definição do *Broker* MQTT

Apesar de funcionais, as estações de irrigação necessitam da escolha e configuração de um servidor para onde possam enviar as informações dos sensores e atuadores disponíveis utilizando o protocolo MQTT, definido no projeto das estações (VOLKMANN, 2022). Por fazer parte da estrutura de aquisição de dados, a definição desse servidor deve levar em conta os critérios desejados pela arquitetura desenvolvida neste trabalho e as limitações e consequências impostas pela distribuição espacial das estações (não se encontram conectadas em uma mesma rede local).

Devido à possibilidade de implementar diferentes tipos de servidores, desde implementações manuais e locais até sistemas gerenciados por serviços na nuvem, foi desenvolvida uma tabela comparativa das propriedades mais relevantes para a tomada de decisão de qual sistema utilizar. As propriedades e os servidores disponíveis são apresentados e comparados na Tabela 2.

Tabela 2: *Comparação entre Brokers MQTT disponíveis para uso*

Broker	**Amazon MSK**	**Amazon MQ**	**AWS IoT com S3**
Linguagem	Scala/Java (nativo)	Múltiplas	Múltiplas
Escalabilidade	Horizontal e Vertical	Horizontal com possível perda de performance (ESTRADA; ASTUDILLO, 2015)	Horizontal
Custo	$$$	$	$
Armazenamento	Como Broker	Como Broker	Como Arquivos

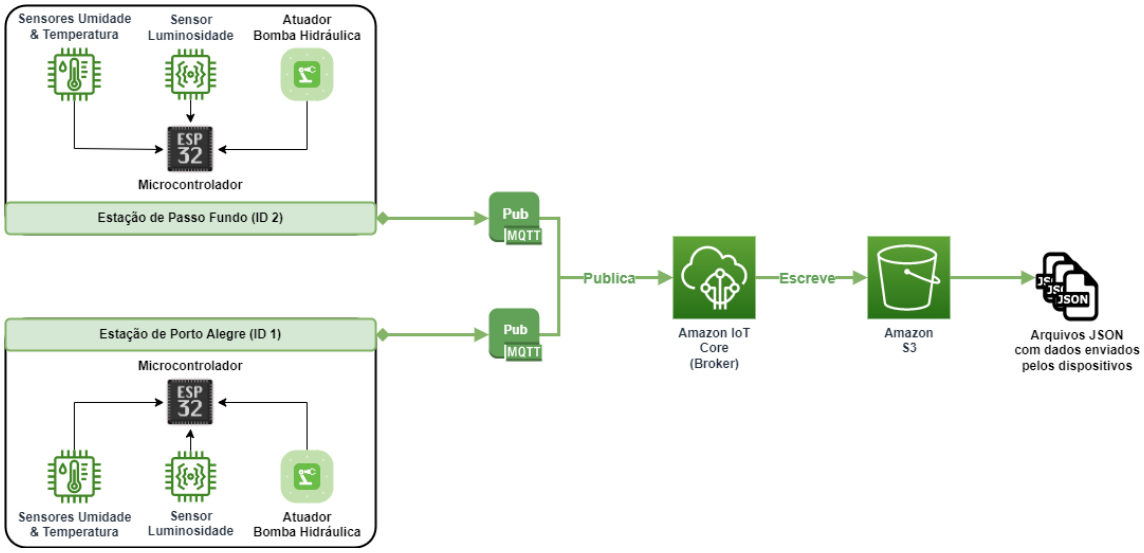
Fonte: O Autor.

Dado o baixo custo e a integração simplificada com o serviço de armazenamento S3, optou-se pelo uso dos serviços AWS IoT e S3 como *broker* para distribuição e consumo dos dados gerados pelas estações de irrigação.

Ademais, a utilização dos serviços AWS permite não só o uso do servidor de mensagens MQTT, como também o direcionamento automático destas mensagens para o serviço de armazenamento na nuvem Amazon S3, sem a preocupação de transmitir essas informações por fora da rede da AWS.

A Figura 7 apresenta o fluxo de dados final da integração das estações de irrigação automática com o servidor escolhido.

Figura 7: *Fluxo de dados provenientes das estações de irrigação usando AWS IoT e S3.*



Fonte: O autor.

3.3 Definição das Estruturas e Fluxos de Dados

O processamento de dados é constituído de várias etapas que vão desde a aquisição dos dados até a agregação destes para serem consumidos por inúmeros processos dependentes.

Como o objetivo deste trabalho é definir uma arquitetura capaz de automatizar os diferentes processos existentes nessas etapas de maneira escalonável, as estruturas que serão utilizadas para permitir o fluxo de dados da fonte até o consumidor final devem ser bem definidas.

Para auxiliar na definição dessas estruturas, o conceito de *Thermal Data* pode ser utilizado. Essa definição trata de categorizar dados em termos de “temperatura" de acordo com a frequência e rapidez necessários para o acesso e consumo dos mesmos. O conceito foi apresentado no evento tecnológico *re:Invent 2018 Big Data Analytics Architectural Patterns & Best Practices*, organizado anualmente pela Amazon, que trouxe formalmente a relação entre a rapidez e a frequência de acesso a um dado com o custo associado.

3.3.1 Estruturas de dados

No problema apresentado neste trabalho, é necessário consumir dados de estações meteorológicas e de irrigação que fornecem dados em formatos, frequências e protocolos distintos. Sendo assim, a primeira estrutura necessária deverá ser capaz de centralizar dados não estruturados (sem formato definido/organizado) de maneira eficiente.

Como, usualmente, o processamento de dados não é um problema fechado, sendo possível incluir e escalar drasticamente o tamanho necessário para armazenamento de inúmeras fontes de dados, optou-se pelo uso de um *bucket* S3 como armazenamento centralizado, o qual possui ótima escalabilidade e elasticidade para resolver o problema apresentado.

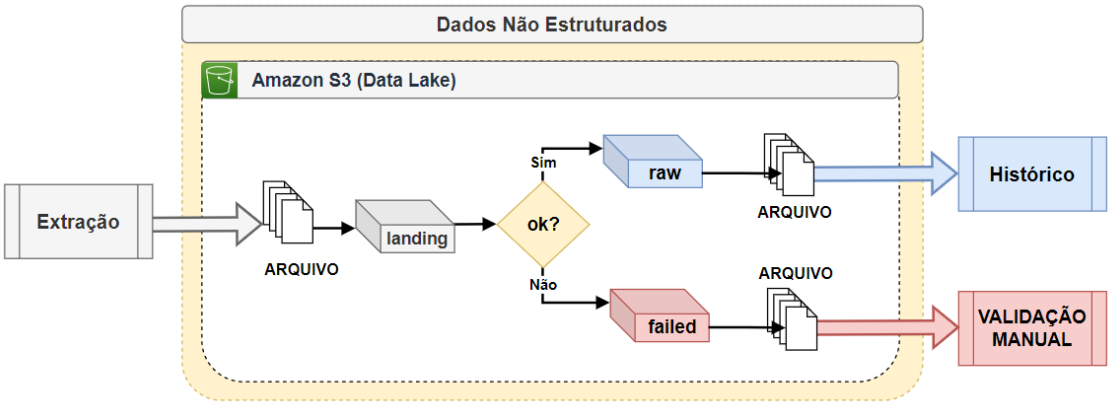
A Figura 8 apresenta a estrutura do *bucket* em questão, que é denominado de *Data Lake*, que é constituído de três áreas principais para armazenamento de dados:

- ***Landing***: área destinada a dados extraídos de suas fontes originais, que não foram alterados por nenhum processo do fluxo de dados. Essa área é, normalmente, subdividida pela natureza dos dados, como INMET e *Volkmann*, referindo-se aos dados provenientes das estações meteorológicas e de irrigação, respectivamente.

- ***Failed***: área destinada a dados com problemas de processamento, sendo estes apenas uma cópia dos dados originais da *landing* para serem processados/corrigidos manualmente pelos profissionais responsáveis pelo monitoramento dos fluxos de dados. Como é usual que falhas no carregamento ocorram por mudanças na estrutura dos arquivos desejados ou até mesmo o corrompimento dos mesmos, é necessária a intervenção manual para a correção dos problemas, motivo pelo qual os arquivos são armazenados em uma área especial.

- ***Raw***: área destinada aos dados já processados da *landing*, evitando, assim, o acúmulo dos mesmos na área anterior. Aqui já é possível ter alguma manipulação do arquivo original porém são alterações que não impactam na informação original, por isso o nome "*raw*", referindo-se a dados "crus" para consumo (sem tratamento).

Figura 8: *Estrutura de armazenamento de dados não estruturados (Data Lake) no S3.*



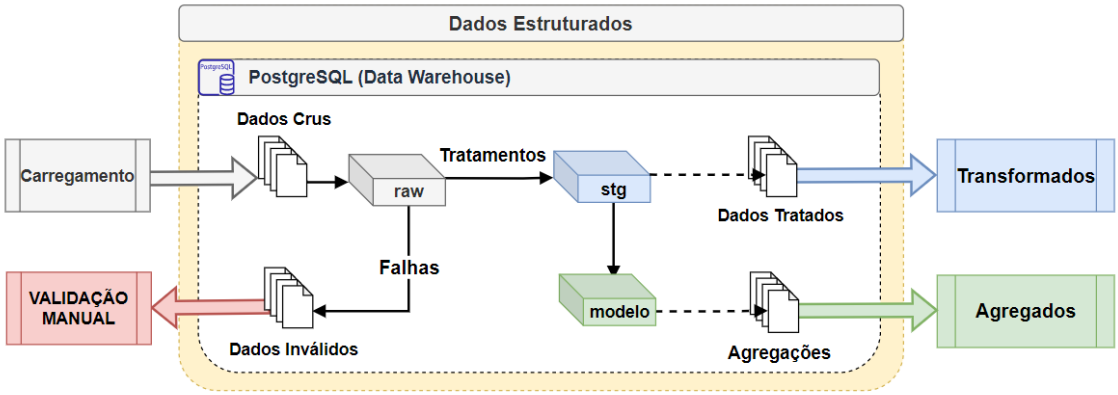
Fonte: O autor.

O uso do Data Lake reduz os custos de manutenção dos dados não estruturados, permitindo assim a existência de um histórico de dados (localizado na área *raw*) que pode ser utilizado por times de ciência de dados para testar hipóteses de maneira eventual. Porém, o Data Lake, como apresentado, não possui um rápido acesso dos dados armazenados, não sendo suficiente para consumidores de alta frequência como APIs ou estruturas *online*.

Sendo assim, dados estruturados serão armazenados em um banco de dados relacional, nomeado *Data Warehouse*, o qual será responsável pelo gerenciamento e fornecimento dos dados “quentes" e estruturados descritos anteriormente. A Figura 9 demonstra a organização que, assim como no Data Lake, utiliza diferentes estruturas responsáveis por manter e apresentar dados em diferentes etapas de processamento:

- ***raw***: área destinada a dados que se encontrem o mais próximo possível de sua fonte original, sem alterações de qualquer natureza sempre que possível, permitindo que processos que utilizam esses dados sejam idempotentes. Dados extraídos são inseridos aqui antes de serem tratados ou agregados a outras fontes. Erros de carregamento também podem ser inseridos em tabelas específicas para garantir um melhor monitoramento dos processos (que podem mudar com o tempo, como, por exemplo, a alteração no modelo de dados enviados pelas fontes).
- ***stg***: abreviação de “*staging*", termo do inglês utilizado para definir algo que está pronto para “entrar em cena", *stg* é a área destinada à dados devidamente tratados originados em *raw*. Nessa área são feitas transformações de dados que evitem duplicidade, que garantam a padronização de unidades e o tratamento de dados inválidos, adicionando uma camada extra de segurança e confiabilidade para quem consome estas informações.
- ***modelo***: área destinada aos dados tratados e agregados. Com a demanda e o surgimento de diferentes tipos de agregação de dados de diferentes fontes, inúmeros modelos podem ser criados para consumo, portanto a nomeação desta área se deu de forma genérica apenas para explicitar que se trata de um ambiente de dados destinados para consumo especializado, como relatórios, monitoramento ou até mesmo treinamento de sistemas de inteligência computacional.

Figura 9: *Estrutura de armazenamento de dados estruturados (Data Warehouse) com Postgres.*



Fonte: O autor.

3.3.2 Fluxo de dados

Com as estruturas definidas, falta então delimitar os processos que farão com que os dados sigam de suas fontes para as estruturas de consumo descritas anteriormente.

Primeiramente, os dados deverão ser obtidos de suas respectivas fontes e transportados até a estrutura de centralização de dados não-estruturados. Assim, fica definida a necessidade de processos de extração de dados como parte inicial do fluxo de dados, saindo da fonte até o *Data Lake*, na área de *landing* descrita anteriormente.

Após a extração, é necessário então processar o conteúdo obtido para que o mesmo seja tratado e carregado no *Data Warehouse*, de onde será devidamente consumido. Porém, dada a necessidade de uma arquitetura escalonável, onde a quantidade de dados processados pode variar com o tempo, torna-se muito custoso utilizar ferramentas computacionalmente isoladas para tratar, transformar e agregar dados de diferentes origens.

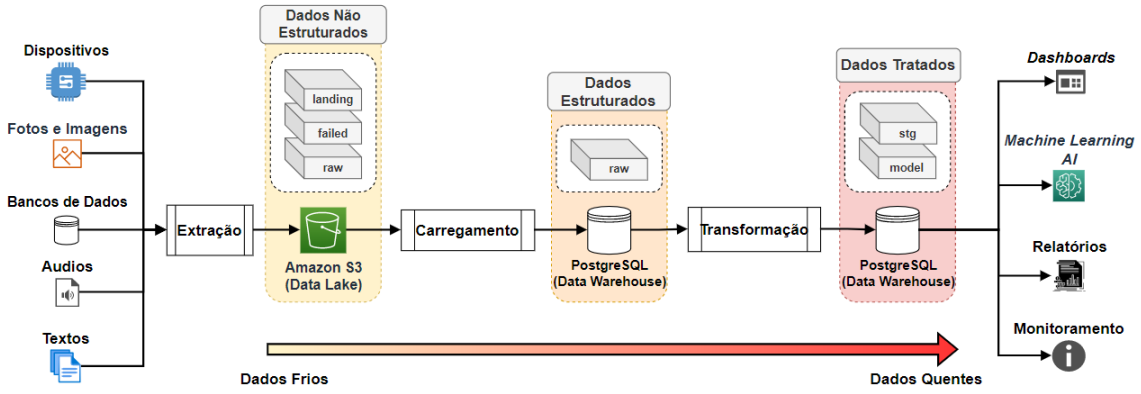
Assim, é preferível que os dados sejam primeiramente carregados no *Data Warehouse* para que sejam processados pelo próprio banco relacional, evitando assim que custos adicionais de ferramentas e processamento de dados sejam assumidos. Ademais, como descrito anteriormente, para evitar que o histórico de arquivos seja reprocessado na área *landing* do *Data Lake*, dados que forem carregados deverão ser movidos para a área *raw* do *Data Lake* e inseridos na área *raw* do *Data Warehouse*.

Por fim, os dados estruturados e devidamente tratados localizados nas áreas *raw* e *stg* do *Data Warehouse* deverão ser agregados em um modelo de dados específico na área *modelo* do *Data Warehouse*, permitindo o consumo especializado dos mesmos e finalizando o fluxo de dados necessário neste trabalho.

Este tipo de abordagem, onde os dados são processados dentro do banco de dados analítico, é conhecida como modelo *ELT* (do inglês *Extract, Load and Transform*) e apresenta cada vez mais benefícios em relação ao modelo *ETL* (do inglês *Extract, Transform and Load*) onde os dados são processados antes de serem carregados no banco de dados (SIVABALAN; MINU, 2021).

A Figura 10 apresenta o fluxo idealizado partindo de fontes genéricas de dados de diferentes naturezas, passando pelo *Data Lake* e, por fim, sendo inseridos no *Data Warehouse* com as devidas transformações. Ainda, a Figura 10 também apresenta o conceito de *thermal data* ao longo do fluxo, permitindo visualizar a consequência de custos das estruturas escolhidas.

Figura 10: *Fluxo de dados seguindo o modelo ELT de processamento.*



Fonte: O autor.

3.4 Definição do Banco de Dados

Como parte do objetivo deste trabalho é apresentar uma arquitetura de dados centralizada, a agregação e armazenamento dos dados desejados deverá ser feita através de um banco de dados, permitindo assim que sistemas de distintas naturezas de implementação (como é o caso das estações do INMET e das estações de irrigação) possam ser relacionados e consumidos sem o impacto dessas diferenças.

Primeiramente, o banco de dados desejado foi avaliado como um banco relacional, onde os dados são estruturados em tabelas e a informação final pode conter fontes de diferentes tabelas, comumente relacionadas entre si por identificadores-chave. Esse tipo de banco é muito utilizado em ambientes de análise e estruturação de dados dada a convenção da linguagem SQL e de suas propriedades de relação e tratamento de operações (PHIRI; KUNDA, 2017).

Ainda, o sistema utilizado será provisionado pelos serviços em nuvem da AWS, facilitando a implementação da comunicação do banco com mecanismos de segurança definidos dentro da nuvem, além de não necessitar de gerenciamento de infraestrutura, permitindo uma melhor escalabilidade da ferramenta.

A Tabela 3 apresenta uma comparação de propriedades relevantes de diferentes sistemas de banco de dados relacionais disponíveis no serviço de banco de dados relacional RDS da AWS, além da solução de armazém de dados Redshift, também da AWS. As estimativas de custo foram calculadas partindo da menor instância disponível para operações genéricas de dados através da calculadora de custos de serviço disponibilizada pela própria AWS.

Tabela 3: *Comparação entre bancos de dados relacionais na AWS*

DB	RDS PostgreSQL	RDS MySQL	AWS Redshift
Implementação	Gerenciado	Gerenciado	Gerenciado
Otimização	Analítica	Transacional	Analítica
Custo	$	$	$$$
Armazenamento	Orientado por Linhas	Orientado por Linhas	Orientado por Colunas
Período Gratuito	12 meses	12 meses	3 meses

Fonte: O Autor.

Dado seu baixo custo, período gratuito ampliado e otimização analítica (importante para o uso como centro de transformação e agregação de dados), foi escolhido o serviço RDS PostgreSQL para ser utilizado como banco de dados relacional do projeto.

Contudo, é importante destacar que, apesar de mais caro, o serviço AWS Redshift seria o mais indicado para este tipo de função em ambientes com maior demanda de processamento de dados, visto que seu armazenamento colunar permite um melhor desempenho em ambientes de alto volume de informações para operações analíticas (BHAGAT; GOPAL, 2012), justificando o custo envolvido na adoção da ferramenta.

3.5 Definição da Estrutura do Apache Airflow

A definição da estrutura do orquestrador é uma etapa crucial para garantir a escalabilidade da arquitetura, visto que ele é responsável por gerenciar a execução de todas as

tarefas de processamento de dados necessárias.

Como o orquestrador é uma ferramenta composta de diversos módulos com responsabilidades distintas, é possível concluir que apenas alguns deles serão expostos ao aumento de demanda com o aumento do número de processos em execução. Isso é confirmado ao analisar o fluxo de operações para a execução de uma tarefa, o qual indica que o processamento ocorre dentro de um *worker* e, portanto, este é o módulo de maior consumo computacional.

Ainda, baseando-se na natureza automática do orquestrador, onde não há dependência de interferência humana constante, é possível perceber que o servidor *web* é o módulo de demanda previsível na arquitetura, visto que é consumido apenas pelos profissionais responsáveis por monitorar o orquestrador. Sendo assim, esse pode ser considerado o módulo de menor demanda na arquitetura e pode ser associado a recursos computacionais estáticos e pré-definidos.

Por fim, o *scheduler*, considerado a peça central do orquestrador, na sua operação convencional, é responsável tanto pela execução (*executor*) das tarefas assim como seu agendamento e verificação de estado. Este trabalho utilizará um módulo *Python* de terceiros (ELZEINY, 2020) cujo objetivo é permitir que o *scheduler* seja responsável apenas pelo agendamento e gerenciamento de estado das tarefas, e a execução das mesmas seja repassada para instâncias executadas pelo *AWS Fargate*.

Com a separação entre *scheduler* e *executor*, torna-se possível considerar o primeiro como um módulo de menor demanda computacional e também associá-lo a recursos computacionais estáticos e pré-definidos. Ainda, o uso do *AWS Fargate* para execução de tarefas permite a conteinerização das mesmas, garantindo assim a previsibilidade de custos e a redução de conflitos de execução.

O contêiner utilizado pela tarefa deverá ser definido e disponibilizado no serviço *AWS ECR*, permitindo assim que tarefas distintas e especializadas possam ser executadas em ambientes adequados. Esse processo ainda adiciona uma camada extra de segurança à execução das tarefas, visto que o contêiner deverá ter as permissões necessárias para acessar os serviços desejados, sendo possível garantir que uma tarefa não tenha permissões para fazer algo que não seja destinada.

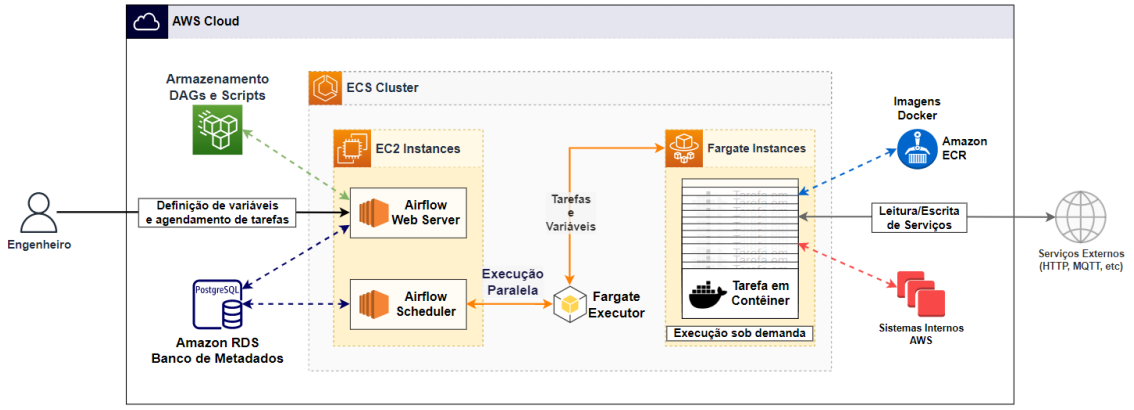
Um benefício muito importante desta estrutura é que as unidades de maior poder computacional, os *workers*, podem ser iniciadas apenas sob demanda. Por se tratar de um orquestrador de tarefas agendadas, muitas vezes os módulos podem ficar ociosos dada a ausência de atividade, gerando desperdício de recursos. O uso do Fargate garante que os contêineres terão seu ciclo de vida iniciado junto da tarefa e encerrado ao fim da mesma, evitando custos desnecessários.

A Figura 11 apresenta a estrutura final e o fluxo de operações para definição e execução de tarefas utilizando o Apache Airflow como orquestrador.

Para fornecer um processo ao orquestrador, o profissional responsável deverá implementar o código padrão de um DAG, apresentado no Apêndice A, e utilizar operadores *Python* (identificados no código como *PythonOperator*) para definição de tarefas. Como boa prática de desenvolvimento, devem ser utilizadas variáveis de execução para que cada tarefa seja idempotente e possa ser gerenciada por um contêiner isolado sem comprometer seu funcionamento.

Como o código para execução deve ser compartilhado entre os contêineres e os módulos *web server* e *scheduler* do Airflow, eles serão alocados em um *bucket* S3 específico que poderá ser acessado por todos os módulos envolvidos na execução do orquestrador. Essa escolha permite que todos compartilhem dos mesmos blocos de código desenvolvidos para

Figura 11: *Estrutura escalonável para o Apache Airflow na nuvem AWS.*



Fonte: O autor.

a execução e orquestração das tarefas.

É importante destacar que, apesar de a solução proposta ser amplamente escalonável em termos de execução de processos simultâneos, o *AWS Fargate* possui um limite padrão de contêineres executados simultaneamente (atualmente 2000), mas que pode ser aumentado se necessário por meio de solicitação à AWS.

3.6 Processos de Extração

Como apresentado na Seção 3.3.2, a primeira etapa do processamento de dados consiste na extração dos dados das fontes de informação para a unidade de armazenamento na nuvem, o *bucket* S3.

Nesta etapa, também torna-se claro um dos primeiros problemas encontrados no processamento de dados: cada sistema fornece informações através de protocolos, frequências e formatos distintos. Assim, faz-se necessário abordar cada processo de extração de maneira especializada, para garantir que os dados sejam coletados sem perdas ou alterações de conteúdo.

3.6.1 Estações meteorológicas

Por se tratar de um sistema que disponibiliza dados meteorológicos utilizando o protocolo HTTP, por meio de uma API, primeiramente faz-se necessário conhecer o formato da requisição e o que é retornado para consumo.

Através do *site* do INMET é possível obter as informações relacionadas à requisição HTTP, a qual consome um método *GET* ao informar o intervalo de datas desejado (ano-mês-dia) assim como o código da estação (A801 para Porto Alegre).

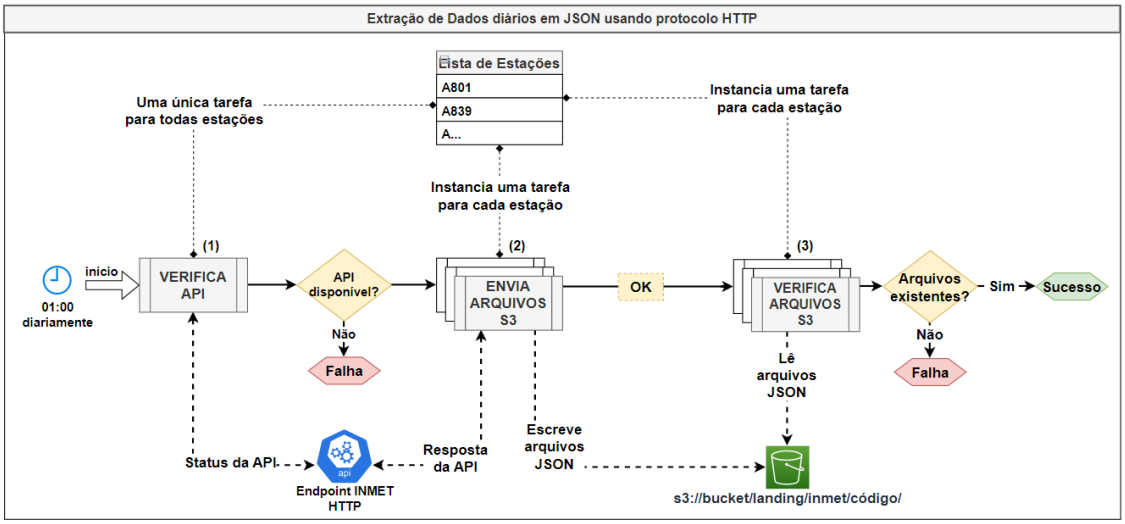
O retorno da API se dá no formato de lista de objetos JSON, com dados meteorológicos de hora em hora para os dias solicitados, e pode ser melhor visualizado no Apêndice B.1. É importante ressaltar que, conforme informado pelo INMET no site da ferramenta, os horários estão no formato de Tempo Universal Coordenado (tempo civil) e que os dados são nulos para horários à frente do momento da solicitação.

Cada objeto retornado na requisição, presentes na Figura 28, possui as seguintes

informações:

- **DC_NOME**: nome da estação.
- **PRE_INS**: pressão atmosférica instantânea, em hPa.
- **TEM_SEN**: sensação térmica, em graus Celsius (^{o}C).
- **VL_LATITUDE**: latitude da estação, em graus (^{o}).
- **PRE_MAX**: pressão atmosférica máxima, em hectopascal (hPa).
- **UF**: sigla do estado nacional que a estação está localizada.
- **RAD_GLO**: radiação solar, em quilojoule por metro quadrado (kJ/m^2).
- **PTO_INS**: ponto de orvalho instantâneo, em graus Celsius (^{o}C).
- **TEM_MIN**: temperatura mínima, em graus Celsius (^{o}C).
- **VL_LONGITUDE**: longitude da estação, em graus (^{o}).
- **UMD_MIN**: umidade relativa do ar mínima, em porcentagem (%).
- **PTO_MAX**: ponto de orvalho máximo, em graus Celsius (^{o}C).
- **VEN_DIR**: direção do vento, em graus (^{o}).
- **DT_MEDICAO**: data da medição, no formato “ano-mês-dia".
- **CHUVA**: quantidade de chuva instantânea, em milímetros (mm).
- **PRE_MIN**: pressão atmosférica mínima, em hectopascal (hPa).
- **UMD_MAX**: umidade relativa do ar máxima, em porcentagem (%).
- **VEN_VEL**: velocidade do vento instantânea, em metros por segundo (m/s).
- **PTO_MIN**: ponto de orvalho mínimo, em graus Celsius (^{o}).
- **TEM_MAX**: temperatura máxima, em graus Celsius (^{o}C).
- **TEN_BAT**: tensão da bateria da estação, em Volts (V).
- **VEN_RAJ**: velocidade do vento máxima, em metros por segundo (m/s).
- **TEM_CPU**: temperatura do processador da estação, em graus Celsius (^{o}C).
- **TEM_INS**: temperatura instantânea, em graus Celsius (^{o}C).
- **UMD_INS**: umidade relativa do ar instantânea, em porcentagem (%).
- **CD_ESTACAO**: código da estação.
- **HR_MEDICAO**: hora da medição, no formato “horaminuto".

Figura 12: *Processo de extração escalonável de dados meteorológicos através do protocolo HTTP.*

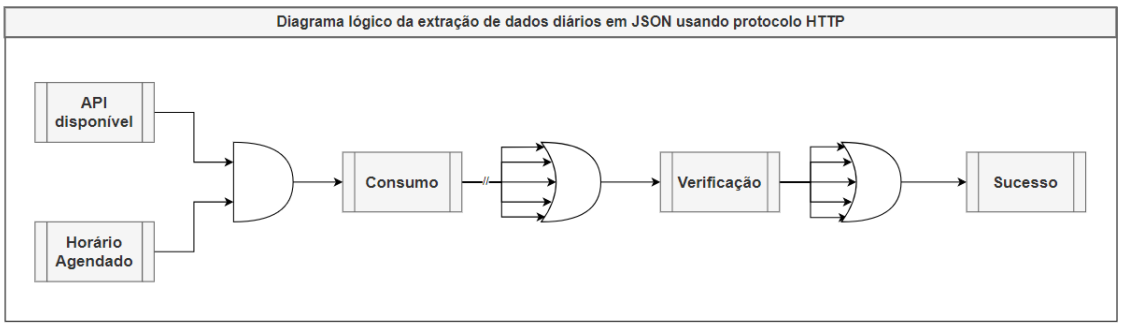


Fonte: O autor.

Portanto, dadas as características da requisição e da resposta obtida, o processo de extração é feito através da escrita dos dados retornados pela API em um *bucket* S3, na área de *landing* descrita na Seção 3.3.2. A Figura 12 detalha as etapas do processo de extração escalonável para uma ou mais estações meteorológicas desejadas.

O comportamento lógico do fluxo de tarefas é dado pela Figura 13, onde as tarefas seguintes às tarefas escalonáveis são tratadas como operações lógicas do tipo “OU", onde o DAG só seguirá no caso da execução bem sucedida de ao menos uma tarefa de extração.

Figura 13: *Diagrama lógico de um DAG genérico de extração escalonável de arquivos por protocolo HTTP.*

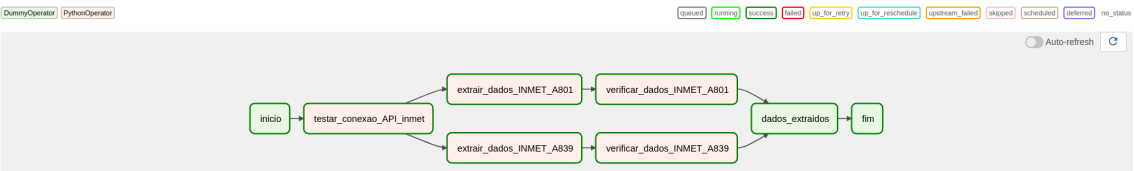


Fonte: O autor.

A Figura 14 apresenta a captura de tela do Airflow após a execução do DAG desenvolvido para orquestrar a execução das etapas descritas anteriormente. O DAG foi definido para ser executado diariamente à 1 hora da manhã e ser capaz de recuperar execuções perdidas para garantir que os dados meteorológicos diários sejam contínuos.

Para ser possível testar a escalabilidade da arquitetura, as tarefas de consumo e escrita dos dados em arquivos JSON são geradas dinamicamente por meio de uma lista de códigos

Figura 14: *Captura de tela do Airflow da representação gráfica do DAG de extração de dados meteorológicos.*



Fonte: O autor.

de estações definida no DAG. Esse comportamento escalonável reflete a proposta de criação e execução isolada de tarefas em paralelo para cada estação, como apresentado no item (2) da Figura 12. O código referente ao DAG está presente no Apêndice A.

3.6.2 Estações de irrigação

As estações disponíveis estão localizadas em cidades distintas e permitem a conexão com um *broker* MQTT para a entrega dos valores de sensores e atuadores disponíveis a cada 5 minutos. Essas características (frequência, meio e protocolo de comunicação) identificam a necessidade de uma abordagem diferente da apresentada para as estações meteorológicas.

Por padrão, a mensagem emitida pela estação contém uma lista de elementos com as seguintes informações:

- **CLIENTE**: código de identificação do cliente.
- **ID**: código de identificação da estação.
- **TIPO**: identificação do tipo de dispositivo, sensor ou atuador.
- **NOME**: nome de identificação do dispositivo.
- **VALOR**: valor instantâneo do sinal do dispositivo.
- **LATITUDE**: latitude da estação, em graus (^{o}).
- **LONGITUDE**: longitude da estação, em graus (^{o}).
- ***DATETIME***: data da medição, no formato “ano-mês-dia hora:minuto:segundo".

Os valores publicados por cada estação compreendem os seguintes dispositivos e unidades:

- **umidade**: sensor de umidade relativa do solo, valor em porcentagem (%).
- **luminosidade**: sensor de luminosidade, valor em milivolts (mV).
- **bomba**: atuador de irrigação, valor lógico (ligado, 1, ou desligado, 0).

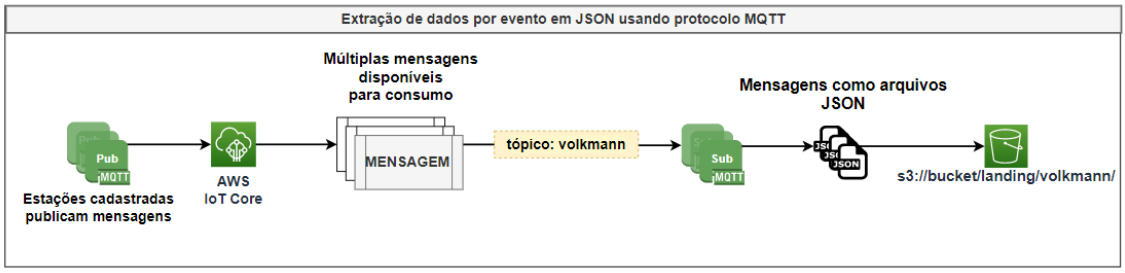
Como descrito na Seção 3.2, o uso do serviço de *broker* gerenciado da AWS, o IoT Core, em conjunto com o *Data Lake* definido na Seção 3.3.2, permitem a extração em tempo real dos dados das estações para o S3.

A configuração dos dispositivos para utilizar o IoT Core, da AWS, é feita de forma manual e necessita que o certificado digital gerado para cada dispositivo adicionado ao *broker* seja adicionado ao código em execução nos microcontroladores. Após essa alteração, basta apontar o endereço do *broker* e a comunicação já configurada nas estações (VOLKMANN, 2022) passa a valer normalmente.

Uma última configuração no IoT Core permite a integração com o *bucket* S3 a partir de políticas internas do serviço, que indicam que cada mensagem recebida pelo *broker* será redirecionada para a área de *landing* no subdiretório “volkmann", destinado aos dados das estações de irrigação.

A Figura 15 apresenta as etapas de extração de dados das estações utilizando o AWS IoT Core e o *Data Lake* no S3, assim como a gravação dos dados em arquivos JSON terminados com a data da mensagem no formato “ano_mês_dia_hora_minuto_segundo".

Figura 15: *Processo de extração escalonável de dados de irrigação por meio do protocolo MQTT.*



Fonte: O autor.

Ainda, desta maneira é possível escalar o número de dispositivos conforme necessário, pois ambos serviços utilizados, IoT Core e S3, são gerenciados pela própria AWS e garantem escalabilidade de infraestrutura necessária para adequar à demanda.

3.7 Processos de Carregamento e Tratamento

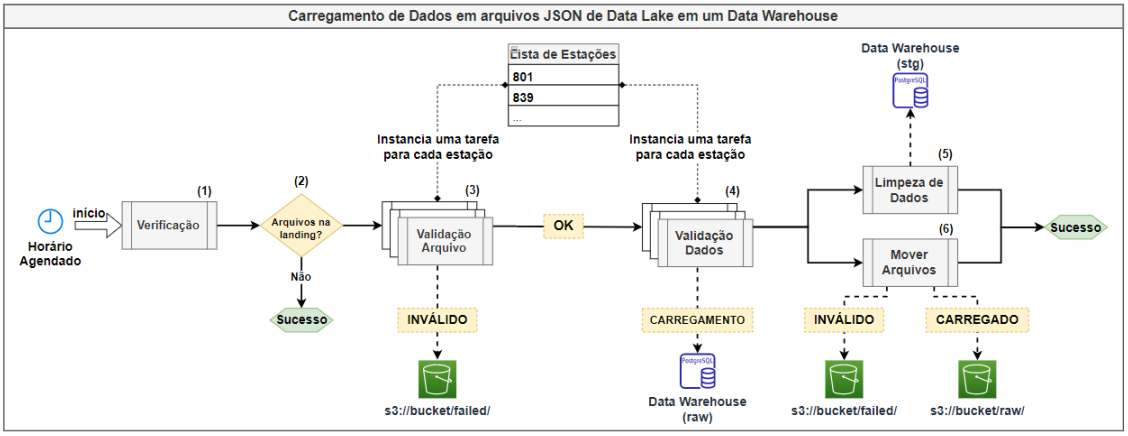
Com os dados presentes no *Data Lake*, agora é necessário carregá-los em um banco de dados analítico para que possam ser devidamente tratados e trabalhados para gerar valor. A abordagem utilizada na extração, de centralizar as informações em um mesmo formato e um mesmo local, permite a simplificação do problema de carregamento visto que agora ambas estações podem ser tratadas de maneira similar, sendo distintas apenas em sua frequência de extração.

Ainda, a escalabilidade do processo agora depende apenas do número e tamanho dos arquivos processados, já que a fonte de informações deixa de ser as estações e passa a ser a área de *landing* do *Data Lake*.

A Figura 16 apresenta as etapas (genéricas) dos processos de carregamento que deverão estar presentes tanto para os dados das estações meteorológicas quanto para os dados das estações de irrigação.

As etapas numeradas na Figura 16 são responsáveis por:

Figura 16: *Processo de carregamento escalonável genérico de arquivos do Data Lake para o Data Warehouse.*



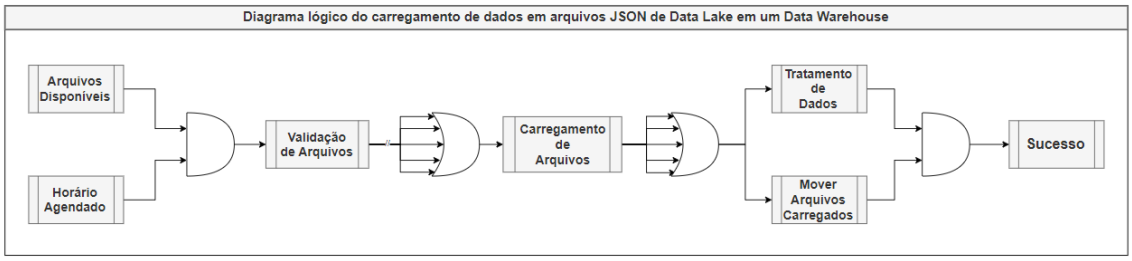
Fonte: O autor.

- **Etapa 1**: acessar a área *landing* do *Data Lake* e verificar se há algum arquivo para ser consumido proveniente das estações listadas.
- **Etapa 2**: caso exista algum arquivo, seguir com a execução do processo, do contrário deve encerrar sem falhas.
- **Etapa 3**: para cada estação, verificar o tamanho e o formato dos arquivos disponíveis para evitar consumir arquivos vazios ou que não estejam no formato JSON. Para o caso de arquivos inválidos, deve mover para a área *failed* do *Data Lake* para que seja analisado manualmente. Ao fim, deve ser enviada uma lista de arquivos válidos para a próxima etapa para que sejam processados e carregados, evitando assim a necessidade de transportar todo o conteúdo dos arquivos em memória.
- **Etapa 4**: para cada estação, acessar a lista de arquivos válidos recebida e, caso houverem arquivos, testar os campos essenciais (que não podem ser nulos). A partir do teste, inserir o dado na tabela correspondente da área *raw* do *Data Warehouse*. Arquivos carregados com sucesso devem ser inseridos em uma lista de arquivos carregados enquanto que arquivos com falha de carregamento devem ser inseridos na lista de falhas, ambas devem ser enviadas para a Etapa 6.
- **Etapa 5**: após o carregamento de todas as estações, os dados presentes na área *raw* do *Data Warehouse* devem ser inseridos na área *stg* estando limpos, padronizados e tendo suas duplicidades resolvidas.
- **Etapa 6**: acessar as listas de arquivos carregados ou falhos e movê-los para suas respectivas áreas no *Data Lake*, permitindo assim que a área de *landing* seja esvaziada para a próxima execução, evitando o reprocessamento de dados.

A Figura 17 apresenta as condições lógicas do fluxo de etapas descrito para os processos de carregamento. Os processos de validação e carregamento, por serem escalonáveis, são tratados como operações lógicas do tipo "OU", onde o DAG só terá sucesso se ao menos uma tarefa de validação e carregamento forem concluídas sem falhas. Dessa maneira, é

possível escalar o número de estações sendo processadas sem alterar o comportamento da solução ou interferir nos processos em andamento.

Figura 17: *Diagrama lógico de um DAG genérico de carregamento escalonável de arquivos.*



Fonte: O autor.

É importante ressaltar também que o tratamento adequado de erros é um passo importante da etapa de carregamento. Aqui, arquivos inválidos são encaminhados para a área de falha do próprio *Data Lake*, enquanto que dados inválidos de arquivos válidos são encaminhados para a área de falha do *Data Warehouse*, permitindo assim uma análise mais sensível aos erros e padrões de falha que ocorrem ao longo da vida útil da solução de forma manual.

3.7.1 Estações meteorológicas

Para os dados provenientes das estações meteorológicas, o processo de carregamento deverá ocorrer pelo menos uma vez ao dia, garantindo assim que todos os dados das estações do dia anterior estejam disponíveis no *Data Warehouse*.

Apesar de também ser possível agendar a tarefa para realizar o carregamento dos dados a cada hora, como para a parte analítica (agregação) do presente trabalho não será necessário apresentar dados em alta frequência ou próximos do tempo real, foi definido o agendamento de execução para três horas após o processo de execução diário.

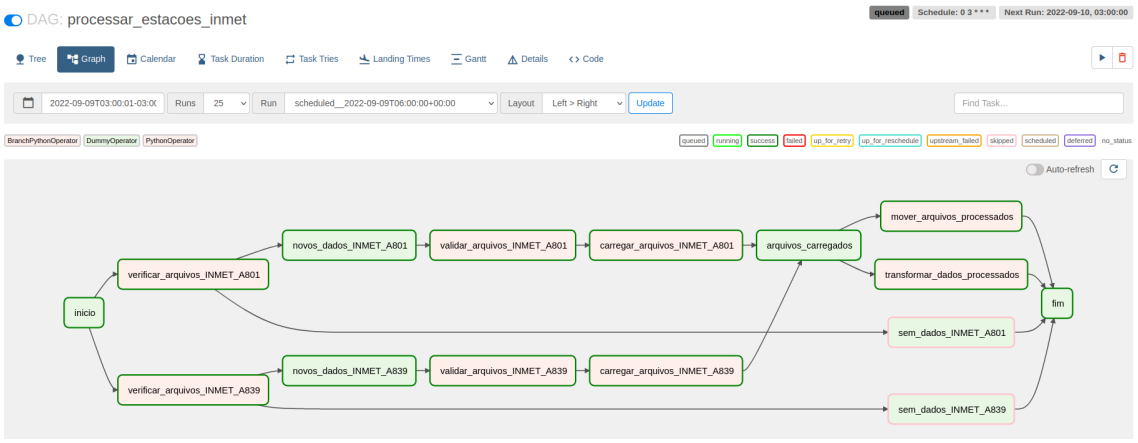
Os dados carregados são definidos em estruturas textuais puras, permitindo assim que qualquer tipo de entrada seja admitido para o campo sendo carregado. A estrutura *raw* utilizada para armazenar os dados meteorológicos é representada visualmente no Apêndice B.2.

A Figura 18 apresenta o DAG final do processo de carregamento das estações meteorológicas após carregar os dados das estações de Porto Alegre e Passo Fundo. Há também a transmissão de listas de arquivos processados de uma tarefa para a tarefa seguinte, evitando assim o processamento de dados na memória interna/banco de dados do orquestrador.

A tarefa de tratamento dos dados carregados, dada por “transformar_dados_carregados"e presente na Figura 18, é responsável pelas conversões e tratamentos utilizando a linguagem de banco de dados “SQL" para a estrutura *stg*. A representação gráfica da relação de transformações utilizadas pode ser encontrada no Apêndice B.2.

A relação entre *raw* e *stg* é, na maior parte de seus elementos, de um-para-um, com exceção dos dados de chuva (convertidos para quantidade e um *booleano*) e de data/horário, unificados em um único campo *data*. Ainda há a presença de um campo extra em cada tabela dado por *atualizacao*, responsável por armazenar o horário em que a linha foi inserida ou atualizada no banco de dados.

Figura 18: *Captura de tela de DAG de carregamento de dados INMET com estações de Passo Fundo e Porto Alegre.*



Fonte: O autor.

3.7.2 Estações de irrigação

O processo de carregamento das estações de irrigação pode seguir, também, as mesmas etapas apresentadas pela Figura 16. Apesar de ter frequência de extração definida por um processo externo ao Airflow, a etapa de verificação de existência de arquivos permite que o processo de carregamento seja desacoplado da frequência de extração, já que apenas irá processar o que estiver disponível na hora de execução.

Sendo assim, o carregamento dos dados de irrigação foi definido para ocorrer de hora em hora, permitindo que não haja o acúmulo excessivo de arquivos para processamento no caso de alteração da frequência de envio das informações das estações para o *broker* MQTT.

Assim como nas estações meteorológicas, os dados são carregados como estruturas textuais na área *raw* do *Data Warehouse* em uma tabela chamada *estacoes_volkmann*, enquanto que os arquivos inválidos são registrados na tabela *err_estacoes_volkmann* na mesma área *raw*. A definição estrutural destas tabelas estão presentes no Anexo B.2.

A Figura 19 apresenta o DAG final do processo de carregamento das estações de irrigação, o qual utiliza os mesmos mecanismos de transmissão de listas de arquivos para inibir o tráfego de dados na memória do orquestrador.

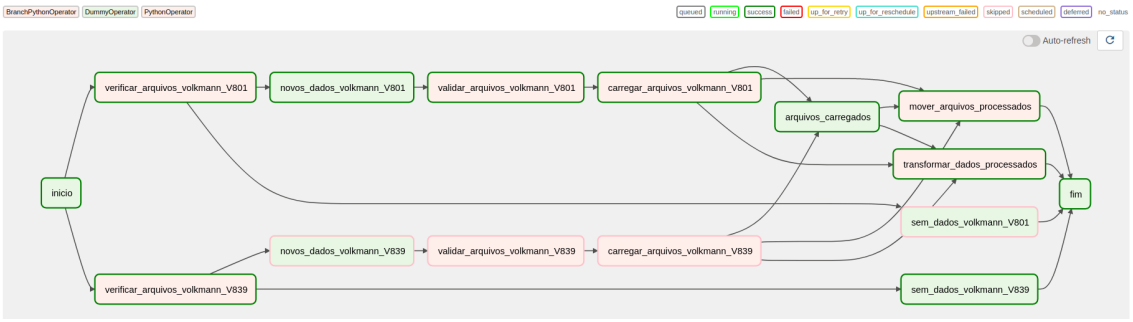
A Figura 20 apresenta a estrutura de dados da área *stg* para as estações de irrigação, obtidos a partir da execução de *scripts* SQL de inserção, presentes no Apêndice A.1.

Dessa maneira, é possível adicionar novas estações de irrigação, adotando códigos de cliente distintos, e as mesmas serão processadas de maneira equivalente e em paralelo sempre que o processo de carregamento for executado.

3.8 Processo de Agregação

Por fim, a agregação de dados apresenta desafios provenientes da parte analítica da solução, onde se faz necessário um modelo capaz de conciliar informações provenientes de todas as fontes de dados envolvidas.

Figura 19: *Captura de tela de DAG de carregamento de dados de irrigação com estações de Passo Fundo (sem dados novos) e Porto Alegre (com dados novos).*



Fonte: O autor.

Figura 20: *Captura de tela de alguns dados presentes na tabela padronizada stg.estacoes_volkmann.*

```
select * from stg.estacoes_volkmann   Enter a SQL expression to filter results (use Ctrl+Space)
```

	cliente	id	tipo	nome	valor	latitude	longitude	datetime
1	V801	1700308	sensor	luminosidade	4,095	-30.030938	-51.216195	2022-04-28 00:00:00.00
2	V801	1700309	sensor	umidade_solo	0.2657	-30.030938	-51.216195	2022-04-28 00:00:00.00
3	V801	1700310	atuador	bomba_hidrau	0	-30.030938	-51.216195	2022-04-28 00:00:00.00
4	V801	1700308	sensor	luminosidade	4,095	-30.030938	-51.216195	2022-04-28 00:05:00.00
5	V801	1700309	sensor	umidade_solo	0.2673	-30.030938	-51.216195	2022-04-28 00:05:00.00
6	V801	1700310	atuador	bomba_hidrau	0	-30.030938	-51.216195	2022-04-28 00:05:00.00
7	V801	1700308	sensor	luminosidade	4,095	-30.030938	-51.216195	2022-04-28 00:10:00.00
8	V801	1700309	sensor	umidade_solo	0.2657	-30.030938	-51.216195	2022-04-28 00:10:00.00
9	V801	1700310	atuador	bomba_hidrau	0	-30.030938	-51.216195	2022-04-28 00:10:00.00
10	V801	1700308	sensor	luminosidade	4,095	-30.030938	-51.216195	2022-04-28 00:15:00.00
11	V801	1700309	sensor	umidade_solo	0.2657	-30.030938	-51.216195	2022-04-28 00:15:00.00
12	V801	1700310	atuador	bomba_hidrau	0	-30.030938	-51.216195	2022-04-28 00:15:00.00
13	V801	1700308	sensor	luminosidade	4,095	-30.030938	-51.216195	2022-04-28 00:20:00.00
14	V801	1700309	sensor	umidade_solo	0.2657	-30.030938	-51.216195	2022-04-28 00:20:00.00

Fonte: O autor.

3.8.1 Modelo de dados

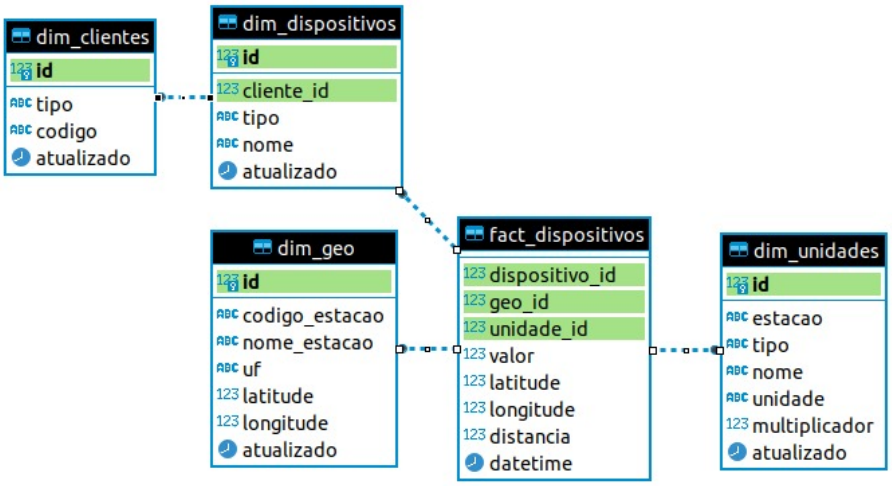
Como exemplo de cenário e objeto de estudo para motivar a agregação dos dados disponíveis, é desejado cruzar as informações provenientes das estações de irrigação com as estações meteorológicas, buscando assim identificar alguma falha de configuração dos sistemas de controle de irrigação e iluminação projetados (VOLKMANN, 2022).

Para o problema apresentado neste trabalho, é possível unir as informações em termos de dados de sensores e atuadores, visto que esta é a principal natureza comum entre os sistemas envolvidos. Sendo assim, a Figura 21 apresenta as estruturas de dados relacionais que constituem o modelo desenvolvido para conciliar os valores das estações disponíveis.

O modelo apresentado é constituído de tabelas *dimensões*, as quais são apresentam valores qualitativos das informações, que são relacionadas com as tabelas *fato*, de natureza quantitativa. As tabelas *dimensão* (dadas pelo prefixo *dim_*) e *fato* (dadas pelo prefixo *fact_*) são:

- **dim_geo**: responsável pelo armazenamento de informações geográficas das estações meteorológicas disponíveis, tais como relação de latitude e longitude com o nome

Figura 21: *Modelo de dados agregados das estações de irrigação e meteorologia com dimensões e fatos.*



Fonte: O autor.

da cidade e o estado federativo da mesma.

- **dim_dispositivos**: responsável pela classificação dos dispositivos disponíveis como sensores ou atuadores, assim como os provedores dos mesmos e suas unidades de medida com base na *dim_unidades*.
- **dim_unidades**: tabela dimensão utilizada para isolar informações de unidades de medidas dos dispositivos. Nesta dimensão é possível definir um valor corretivo para diferentes respostas dos dispositivos adotados, dado pelo campo *multiplicador*, permitindo assim manter a mesma ordem de grandeza dos dados agregados.
- **dim_clientes**: responsável por armazenar a relação de estações cadastradas que provêm dados de irrigação ou dados meteorológicos.
- **fact_dispositivos**: tabela que relaciona as informações qualitativas com os eventos disponíveis dos sensores e atuadores das estações meteorológicas e de irrigação.

A partir destas estruturas, é possível consumir as informações da tabela *fato* e avaliar as configurações de controle de luminosidade e umidade das estações de irrigação em razão dos eventos meteorológicos localizados próximos das mesmas por meio de superposição de eventos.

Aqui, os dados provenientes da área *stg* são transformados em eventos únicos de dispositivos e suas características, permitindo tratar ambos modelos de estações como uma mesma natureza.

Outro grande benefício da separação da informação em tabelas qualitativas e quantitativas é a descrição do conteúdo da tabela fato sem duplicidade de informações, visto que características dos dispositivos são relacionadas por chaves únicas das tabelas dimensão, otimizando tanto a leitura posterior das informações como também o armazenamento de eventos de sensores e atuadores ao longo do tempo.

Por fim, o modelo desenvolvido permite não apenas a integração das estruturas existentes de ambas as estações como também a implementação de alterações físicas nas

mesmas, tais como a alteração ou adição de sensores ou atuadores. Caso novas estações de irrigação sejam desenvolvidas utilizando dispositivos com unidades de medida diferentes das apresentadas neste projeto, é possível incluir essa alteração, e uma possível correção linear, no modelo sem que o consumidor dos dados finais seja afetado.

3.8.2 Regras do modelo agregado

Cada tabela do modelo agregado é construída em cima de “regras de negócio" estabelecidas para o problema sendo analisado. Assim, tanto as dimensões quanto os fatos são especializados no assunto o qual serão utilizados pela busca de respostas, que neste caso é definido como o desejo de avaliar as configurações dos controladores de irrigação e iluminação com base nos dados meteorológicos mais próximos.

As principais regras que devem ser levadas em consideração são as que dizem respeito à relação das estações meteorológicas com as estações de irrigação, portanto é necessário que a relação entre o evento de um dispositivo (*fact_dispositivo*) e um elemento de geolocalização (*dim_geo*) seja definida pela menor distância entre as coordenadas de latitude e longitude do evento e do elemento de geolocalização.

Assim, para encontrar o valor aproximado da distância entre coordenadas dos pontos A e B, e considerando a terra como uma esfera perfeita dada a restrição do problema apenas à localização geográfica brasileira, é possível utilizar a fórmula de Haversine (1) (KIFANA; ABDUROHMAN, 2012) para encontrar a metade do seno verso entre A e B.

$$hav(\Theta) = sin^2(\Delta\psi/2) + cos(\psi_A) \cdot cos(\psi_B) \cdot sin^2(\Delta\lambda/2) \tag{1}$$

A equação consiste em encontrar o valor da metade do seno verso, dado por $hav(\Theta)$, utilizando a latitude do ponto A (ψ_A), a latitude do ponto B (ψ_B), a diferença entre latitudes ($\Delta\psi$) e longitudes ($\Delta\lambda$) dos dois pontos.

Após o cálculo da metade do seno verso entre os dois pontos, dado pela Equação (1), é possível encontrar a distância, em metros, entre eles através da relação trigonométrica apresentada na Equação (2).

$$d = 2 \cdot R \cdot arcsin\left(\sqrt{hav(\Theta)}\right) \tag{2}$$

Onde R é a constante do raio da terra, definido como o valor médio do raio entre os polos terrestres e o raio do equador, resultando no valor de 6378 quilômetros. O valor final da distância d entre os dois pontos, obtido pela Equação (2), é dado em metros e trata-se de uma aproximação da distância real da reta entre as coordenadas na terra, sem considerar elevações de terreno e variações do raio terrestre, sendo suficiente para o problema abordado.

Portanto, fica definida a regra de negócio em que o evento de um dispositivo presente em *fact_dispositivo* será associado à entrada de menor distância das coordenadas do evento em relação à tabela *dim_geo*, dada pela Equação (2).

Assim, durante o processo de atualização dos dados do modelo, o *script* SQL responsável pela inserção dos dados na tabela deverá calcular as distâncias de um evento de dispositivo em relação às estações meteorológicas disponíveis e deverá atribuir a esse evento o identificador da estação meteorológica (*dim_geo*) mais próxima.

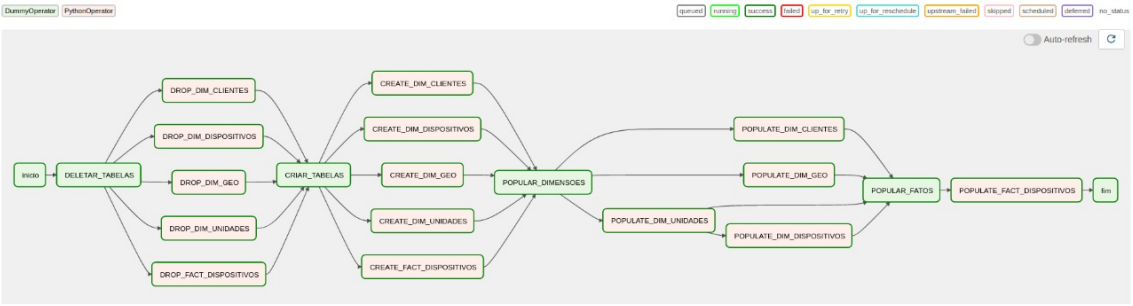
3.8.3 Implementação no Airflow

Como o modelo de dados apresentado é constituído inteiramente de estruturas provenientes do *Data Warehouse*, a automação do processo de atualização dos dados deve ser

feita através do agendamento de tarefas Python para execução de *scripts* SQL, que serão responsáveis por recriar e popular as tabelas dimensão e fato diariamente.

A Figura 22 apresenta o DAG desenvolvido para a atualização do modelo definido utilizando uma tarefa para cada tabela envolvida. Fica também evidente a sequência utilizada para a atualização do modelo, sendo primeiramente executados os códigos referentes às tabelas de dimensão e só então executado o código da tabela fato, respeitando a dependência de informações.

Figura 22: *Captura de tela do Airflow da representação gráfica do DAG de atualização do modelo agregado.*



Fonte: O autor.

Como o objetivo do modelo em questão é monitorar o comportamento do controle definido nas estações de irrigação com base nos dados meteorológicos diários, a frequência de atualização dos valores foi definida como um evento diário executado às 7 horas da manhã, no Tempo Universal Coordenado. Dessa maneira é possível obter os dados mais recentes provenientes das estações meteorológicas e de irrigação antes da atualização do modelo.

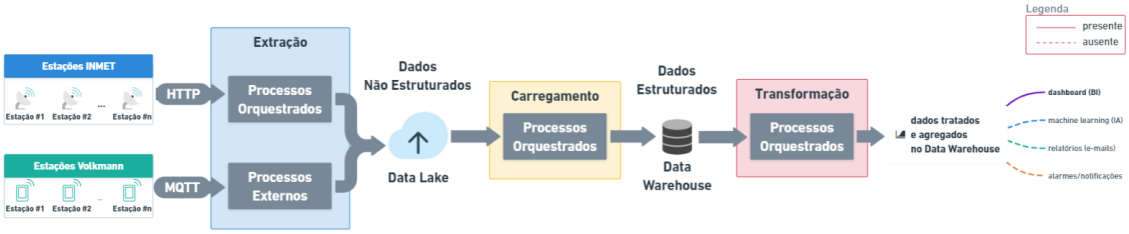
Por fim, diferentemente dos processos de extração e carregamento, o DAG de agregação não depende de nenhuma medida de escalonamento de tarefas dinâmicas, visto que todo o processamento é realizado dentro do *Data Warehouse*.

Os códigos referentes ao DAG e aos *scripts* SQL utilizados podem ser encontrados no Apêndice A.

4 RESULTADOS

A proposta final da arquitetura escalonável na nuvem para processamento de dados de sistemas IoT distribuídos é apresentada na Figura 23, a qual descreve o fluxo dos dados desde as estações utilizadas até o fornecimento dos dados agregados para diferentes fontes de consumo. Para averiguar a capacidade de solução da arquitetura, é possível dividir o problema em duas propriedades: escalonamento e processamento.

Figura 23: *Diagrama final da arquitetura proposta para processamento de dados de sistemas IoT.*



Fonte: O autor.

Para a análise visual das informações presentes no *Data Warehouse* foi utilizada a ferramenta de visualização de dados *Redash*, que permite o desenvolvimento de gráficos a partir de consultas a um banco de dados.

4.1 Análise de Escalonamento

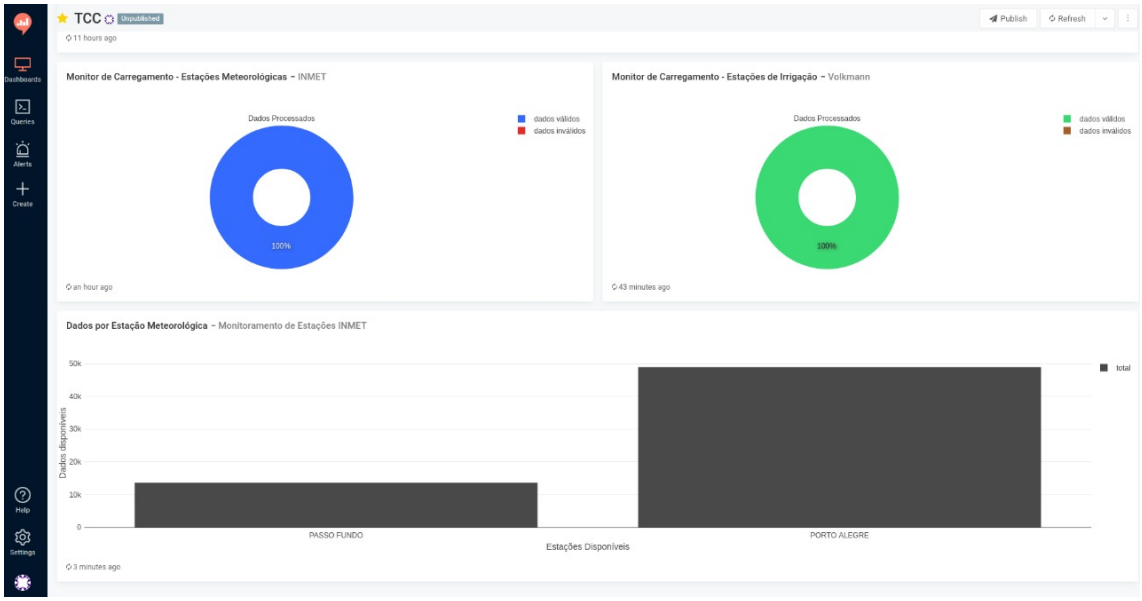
Para a avaliação da capacidade de escalonamento da arquitetura proposta, esta deve ser capaz de receber um número maior de requisições sem apresentar problemas de processamento como gargalos ou falhas de concorrência por paralelismo.

O principal problema e sintoma em uma execução não escalonável que tem seus processos expandidos é o comportamento errático da solução onde tarefas competem pelos recursos disponíveis das máquinas que executam os códigos definidos, gerando falhas operacionais ou até mesmo a perda (ou poluição) de dados. Esse problema deve ser gerenciado pelo próprio orquestrador, o qual precisa distribuir tarefas aos *workers* (e na implementação atual, solicitar a criação dos mesmos) assim que a tarefa tenha de ser executada.

Para testar este funcionamento, foram utilizadas duas listas de estações meteorológicas a serem processadas, onde a primeira continha apenas duas estações (Porto Alegre e Passo Fundo) e a segunda sete vezes mais. A Figura 24 apresenta gráficos de monitoramento das

operações de carregamento e a presença de apenas duas estações meteorológicas no *Data Warehouse*.

Figura 24: *Dashboard de monitoramento com duas estações meteorológicas disponíveis.*



Fonte: O autor.

Com um número maior de estações sendo consumidas, o orquestrador foi capaz de atribuir cada tarefa à uma instância Fargate correspondente sem sofrer perda de dados e sem falhas computacionais, que puderam ser aferidas através do estado íntegro dos dados ingeridos e do aumento do número de estações meteorológicas disponíveis, apresentado na Figura 25.

Capturas de tela dos DAGs de extração e processamento com um maior número de estações meteorológicas definidas podem ser encontradas no Apêndice B.4.

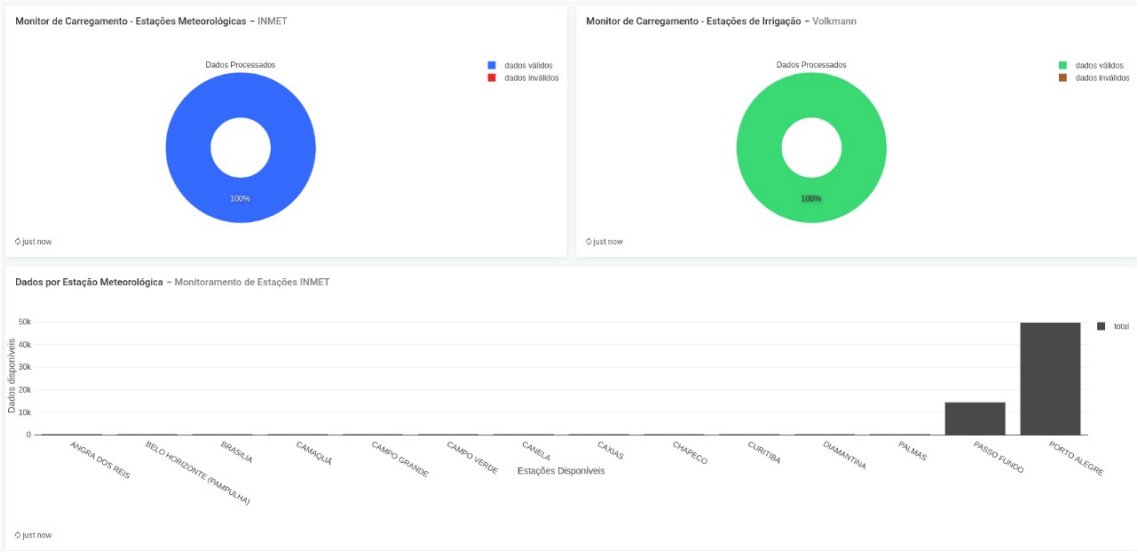
Um contraponto relevante para o comportamento escalonável das tarefas na arquitetura proposta se dá na necessidade de desenvolver os DAGs levando em consideração maneiras de alterar o número de tarefas de forma dinâmica. Essa característica, apesar de funcional, adiciona complexidade no desenvolvimento das tarefas de processamento de dados que deve ser levado em consideração em uma aplicação real da arquitetura.

A análise por qualidade de dados foi feita de forma inicial e prática para assegurar a qualidade da análise de processamento, porém, a análise do tempo de execução de cada tarefa executada poderia ser utilizada para avaliar a evolução da escalabilidade da solução ao longo do tempo.

4.2 Análise de Processamento

A capacidade de processamento de dados da arquitetura deve permitir consumir e agregar os dados de estações de irrigação e meteorológicas sem que a inclusão ou remoção de estações interfira nestes processos. Ou seja, ao consumir os dados de todas as estações meteorológicas disponíveis, a solução apresentada deve ser capaz de agregar dados de estações de irrigação com as estações meteorológicas adequadas baseando-se na regra

Figura 25: *Dashboard de monitoramento com várias estações meteorológicas disponíveis.*



Fonte: O autor.

estabelecida na Seção 3.8.

Para a validação da capacidade de processamento dos dados sem depender da quantidade de estruturas envolvidas, ao fim da atualização do modelo analítico desenvolvido, os eventos da estação de irrigação de Porto Alegre, assim como os eventos da respectiva estação meteorológica, devem estar ambos associados à mesma geolocalização da estação meteorológica de Porto Alegre.

A demonstração da funcionalidade de associação de eventos de dispositivos com a estação meteorológica mais próxima pode ser encontrada no Apêndice B.3.

Ainda, como consequência do correto cruzamento de dados de ambas as estações, é possível responder às perguntas apresentada na Seção 3.8 sobre a configuração do sistema de irrigação e seus controladores. A Figura 26 apresenta os eventos de ambas estações em um período de 7 dias, onde é possível observar que eventos meteorológicos, como dias de chuva ou de baixa radiação solar, poderiam acarretar na incorreta ativação da iluminação artificial caso o controlador fosse projetado com um ponto de ativação muito inferior ao utilizado, de 3000 mV (VOLKMANN, 2022).

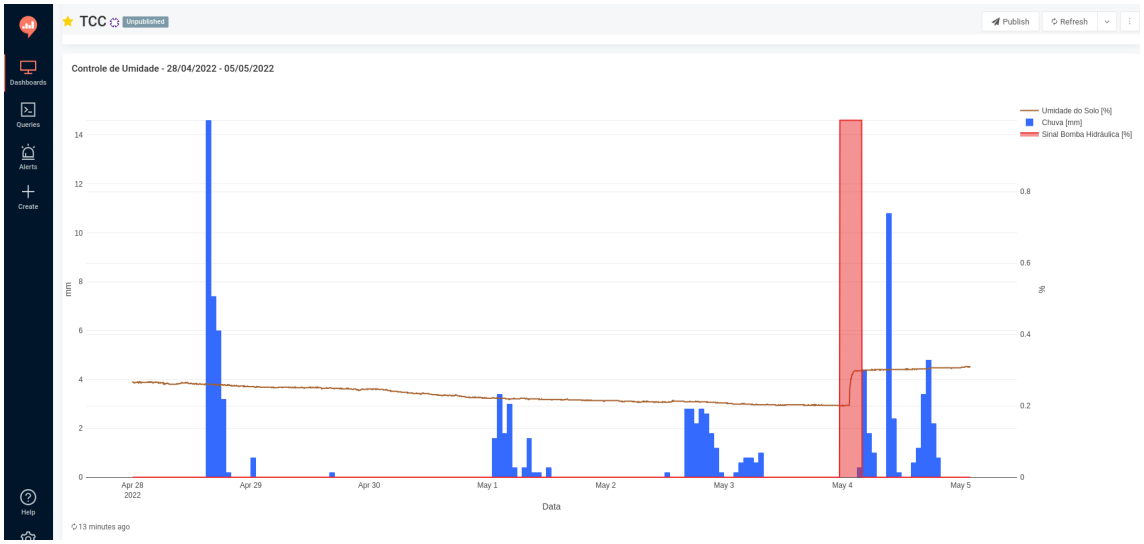
Também é possível avaliar a correta configuração dos controladores de irrigação na Figura 27, onde os mesmos acionam as bombas hidráulicas de maneira adequada, mantendo a umidade relativa do solo nos intervalos desejados pelo projeto do controlador (VOLKMANN, 2022), mesmo com a interferência de dias chuvosos, garantindo assim que os sensores de umidade do solo não estão expostos a ponto de terem suas leituras comprometidas por eventos naturais.

Porém, é importante ressaltar que a arquitetura apresentada, mesmo sendo capaz de seguir adequadamente as regras e algoritmos de processamento de dados com alteração no número de fontes utilizadas, não leva em consideração a adição de novas fontes de dados, sendo necessário seguir as mesmas etapas apresentadas nas Seções 3.6 e 3.7 para o desenvolvimento de DAGs responsáveis pelas novas fontes.

Figura 26: *Gráfico do comportamento do sinal de luminosidade frente à condições climáticas.*

Fonte: O autor.

Figura 27: *Gráfico do comportamento do controlador de umidade frente à condições climáticas.*



Fonte: O autor.

5 CONCLUSÕES

A arquitetura desenvolvida ao longo deste trabalho contou com a implementação integral de seus componentes através de serviços disponíveis na nuvem pela AWS, permitindo não apenas a integração de sistemas adaptados para tarefas específicas (orquestrador Apache Airflow) assim como de sistemas gerenciados (banco de dados relacional e *broker* IoT Core) pela própria AWS para garantir a possibilidade de escalonamento e processamento de tarefas conforme a demanda.

Após as análises dos resultados nos cenários apresentados, conclui-se que a arquitetura desenvolvida é capaz de solucionar o problema apresentado sem que seja necessário alterar seus componentes para a inclusão ou remoção de estações meteorológicas e de irrigação, desde que estas respeitem a mesma estrutura de dados e comportamento que a utilizada no desenvolvimento dos DAGs para cada etapa.

Ainda, a associação de instâncias Fargate para a execução de tarefas permitiu a elasticidade da solução de maneira que não foi necessário arcar com custos de *hardware* ocioso para os *workers*, permitindo um melhor gerenciamento de recursos sem comprometer a capacidade de escalonamento de tarefas.

Também foi possível avaliar a correta configuração dos controladores e da instalação das estações de irrigação projetadas em um trabalho anterior a este, trazendo luz ao valor associado à continuidade dos processos de automação como sistemas geradores de dados e as implicações dos mesmos em um mundo cada vez mais interconectado. Em um caso de aplicação dos sistemas de irrigação desenvolvidos como negócio, a arquitetura apresentada neste trabalho permitiria o monitoramento e validação destes sistemas localizados em diversas regiões do Brasil, centralizando a informação gerada pelos mesmos e permitindo a identificação e resolução de problemas de maneira rápida e assertiva.

Como melhoria e possível ponto de atenção para trabalhos futuros, destaca-se o uso de ferramentas e recursos que podem ser substituídos por soluções completamente gerenciadas pela AWS, como é o caso do próprio Apache Airflow e do *Data Warehouse*, com a adição de custo financeiro para a solução. Apesar de não alterar o funcionamento e comportamento da arquitetura proposta, a escolha das ferramentas está condicionada à realidade do problema abordado, sendo necessário reavaliar as estruturas desejadas em um cenário mais ou menos complexo e de maior ou menor versatilidade de fontes de dados.

De forma similar, é possível também avaliar a utilização de recursos distintos entre outros provedores de serviços em nuvem, permitindo uma versatilidade ainda maior tanto para as ferramentas escolhidas quanto para os custos envolvidos na solução. Porém, é importante levar em consideração que a utilização de diferentes plataformas em nuvem demandam uma maior atenção para com as estruturas de rede e segurança, pois tornam a solução vulnerável à ataques e vazamento de dados.

REFERÊNCIAS

ANDRY, F.; RIDOLFO, R.; HUFFMAN, J. Migrating Healthcare Applications to the Cloud through Containerization and Service Brokering. In: PROCEEDINGS of the International Conference on Health Informatics. [S.l.]: SCITEPRESS - Science, 2015. DOI: 10.5220/0005249601640171. Disponível em: <https://doi.org/10.5220/0005249601640171>.

BALI, M. S. et al. Smart Architectural Framework for Symmetrical Data Offloading in IoT. *Symmetry*, MDPI AG, v. 13, n. 10, p. 1889, out. 2021. DOI: 10.3390/sym13101889. Disponível em: <https://doi.org/10.3390/sym13101889>.

BHAGAT, V.; GOPAL, A. Comparative Study of Row and Column Oriented Database. In: 2012 Fifth International Conference on Emerging Trends in Engineering and Technology. [S.l.: s.n.], 2012. P. 196–201. DOI: 10.1109/ICETET.2012.56.

BRYLINSKI, A.-S.; BHATTACHARJYA, A. Overview of HTTP/2. In: PROCEEDINGS of the Second International Conference on Internet of Things, Data and Cloud Computing. Cambridge, United Kingdom: Association for Computing Machinery, 2017. (ICC '17). ISBN 9781450347747. DOI: 10.1145/3018896.3065841. Disponível em: <https://doi.org/10.1145/3018896.3065841>.

BURKAT, K. et al. Serverless Containers – Rising Viable Approach to Scientific Workflows. In: 2021 IEEE 17th International Conference on eScience (eScience). [S.l.: s.n.], 2021. P. 40–49. DOI: 10.1109/eScience51609.2021.00014.

ELZEINY, A. *Apache Airflow: Native AWS Executors*. [S.l.]: GitHub, 2020. https://github.com/aelzeiny/airflow-aws-executors.

ESTRADA, N.; ASTUDILLO, H. Comparing scalability of message queue system: ZeroMQ vs RabbitMQ. In: 2015 Latin American Computing Conference (CLEI). [S.l.: s.n.], 2015. P. 1–6. DOI: 10.1109/CLEI.2015.7360036.

EUROTECH, I. B. M. C. MQTT specification version 3.1. out. 2010. In: PROCEEDINGS of the 24th WORKSHOP ON REAL-TIME PROGRAMMING. Schloss Dagstuhl, Germany: [s.n.], 1999. P. 9–14. Disponível em: <https://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html. Acesso em: 12 abril 2019.

GRIGORIK, I. *High performance browser networking*. Sebastopol, CA: O'Reilly Media, set. 2013.

GÜNTHER, W. A. et al. Debating big data: A literature review on realizing value from big data. *The Journal of Strategic Information Systems*, Elsevier BV, v. 26, n. 3, p. 191–209, set. 2017. DOI: 10.1016/j.jsis.2017.07.003.

HASSAN, S.; BAHSOON, R.; BUYYA, R. Systematic scalability analysis for microservices granularity adaptation design decisions. *Software: Practice and Experience*, Wiley, v. 52, n. 6, p. 1378–1401, jan. 2022. DOI: 10.1002/spe.3069. Disponível em: <https://doi.org/10.1002/spe.3069>.

HIMESH, S. Digital revolution and Big Data: a new revolution in agriculture. *CAB Reviews: Perspectives in Agriculture, Veterinary Science, Nutrition and Natural Resources*, CABI Publishing, v. 13, n. 021, abr. 2018. DOI: 10.1079/pavsnnr201813021.

KHAN, N. et al. Big Data: Survey, Technologies, Opportunities, and Challenges. *The Scientific World Journal*, Hindawi Limited, v. 2014, p. 1–18, 2014. DOI: 10.1155/2014/712826.

KIFANA, B. D.; ABDUROHMAN, M. Great circle distance methode for improving operational control system based on gps tracking system. *International Journal on Computer Science and Engineering*, Citeseer, v. 4, n. 4, p. 647, 2012.

KUGELE, S.; HETTLER, D.; PETER, J. Data-Centric Communication and Containerization for Future Automotive Software Architectures. In: 2018 IEEE International Conference on Software Architecture (ICSA). [S.l.]: IEEE, abr. 2018. DOI: 10.1109/icsa.2018.00016. Disponível em: <https://doi.org/10.1109/icsa.2018.00016>.

LEHRIG, S.; EIKERLING, H.; BECKER, S. Scalability, Elasticity, and Efficiency in Cloud Computing. In: PROCEEDINGS of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures. [S.l.]: ACM, mai. 2015. DOI: 10.1145/2737182.2737185. Disponível em: <https://doi.org/10.1145/2737182.2737185>.

LI, X.; ZOU, B. *An Automated Data Engineering Pipeline for Anomaly Detection of IoT Sensor Data*. [S.l.]: arXiv, 2021. DOI: 10.48550/ARXIV.2109.13828. Disponível em: <https://arxiv.org/abs/2109.13828>.

MILLER, H. G.; MORK, P. From Data to Decisions: A Value Chain for Big Data. *IT Professional*, v. 15, n. 1, p. 57–59, 2013. DOI: 10.1109/MITP.2013.11.

PHIRI, H.; KUNDA, D. A Comparative Study of NoSQL and Relational Database. *Zambia ICT Journal*, v. 1, p. 1–4, dez. 2017. DOI: 10.33260/zictjournal.v1i1.8.

QUINCOZES, S.; EMILIO, T.; KAZIENKO, J. MQTT Protocol: Fundamentals, Tools and Future Directions. *IEEE Latin America Transactions*, v. 17, n. 09, p. 1439–1448, 2019. DOI: 10.1109/TLA.2019.8931137.

AL-RAKHAMI, M. et al. A lightweight and cost effective edge intelligence architecture based on containerization technology. *World Wide Web*, Springer Science e Business Media LLC, v. 23, n. 2, p. 1341–1360, mai. 2019. DOI: 10.1007/s11280-019-00692-y. Disponível em: <https://doi.org/10.1007/s11280-019-00692-y>.

RAYES, A.; SALAM, S. The Internet in IoT: OSI, TCP/IP, IPv4, IPv6 and Internet Routing. In: INTERNET of Things From Hype to Reality: The Road to Digitization. Cham: Springer International Publishing, 2017. P. 35–56. ISBN 978-3-319-44860-2. DOI: 10.1007/978-3-319-44860-2_2.

ROSE, K.; ELDRIDGE, S. D.; CHAPIN, L. THE INTERNET OF THINGS : AN OVERVIEW Understanding the Issues and Challenges of a More Connected World. In.

SAGGI, M. K.; JAIN, S. A survey towards an integration of big data analytics to big insights for value-creation. *Information Processing &amp Management*, Elsevier BV, v. 54, n. 5, p. 758–790, set. 2018. DOI: 10.1016/j.ipm.2018.01.010. Disponível em: <https://doi.org/10.1016/j.ipm.2018.01.010>.

SINGH, P. Airflow. In: LEARN PySpark. [S.l.]: Apress, 2019. P. 67–84. DOI: 10.1007/978-1-4842-4961-1_4. Disponível em: <https://doi.org/10.1007/978-1-4842-4961-1_4>.

SIVABALAN, S.; MINU, R. I. Heterogeneous Data Integration with ELT and Analytical MPP Database for Data Analysis Application. In: 2021 Innovations in Power and Advanced Computing Technologies (i-PACT). [S.l.: s.n.], 2021. P. 1–5. DOI: 10.1109/i-PACT52855.2021.9696841.

SURBIRYALA, J.; RONG, C. Cloud Computing: History and Overview. In: 2019 IEEE Cloud Summit. [S.l.: s.n.], 2019. P. 1–7. DOI: 10.1109/CloudSummit47114.2019.00007.

VOLKMANN, T. B. *Implantação de sistema IoT para automação residencial e irrigação de jardins.* [S.l.: s.n.], 2022. Disponível em: <https://search.ebscohost.com/login.aspx?direct=true&AuthType=shib&db=cat07377a&AN=sabi.001141941&lang=pt-br&scope=site&authtype=guest,shib&custid=s5837110&groupid=main&profile=eds>.

VUKOVIC, M. Internet Programmable IoT: On the Role of APIs in IoT: The Internet of Things (Ubiquity Symposium). *Ubiquity*, Association for Computing Machinery, New York, NY, USA, v. 2015, November, nov. 2015. DOI: 10.1145/2822873. Disponível em: <https://doi.org/10.1145/2822873>.

YACCHIREMA, D. C.; PALAU, C. E.; ESTEVE, M. Enable IoT interoperability in ambient assisted living: Active and healthy aging scenarios. *2017 14th IEEE Annual Consumer Communications & Networking Conference (CCNC)*, p. 53–58, 2017.

YANG, C. et al. Big Data and cloud computing: innovation opportunities and challenges. *International Journal of Digital Earth*, Informa UK Limited, v. 10, n. 1, p. 13–53, nov. 2016. DOI: 10.1080/17538947.2016.1239771. Disponível em: <https://doi.org/10.1080/17538947.2016.1239771>.

APÊNDICE A - CÓDIGOS DESENVOLVIDOS

A.1 Scripts SQL

Dada a grande quantidade de linhas de código em diferentes linguagens utilizadas ao longo deste trabalho, estes foram centralizados na ferramenta de versionamento de código na nuvem, *Github*.

O repositório contendo todos os códigos utilizados pode ser encontrado através da seguinte URL:

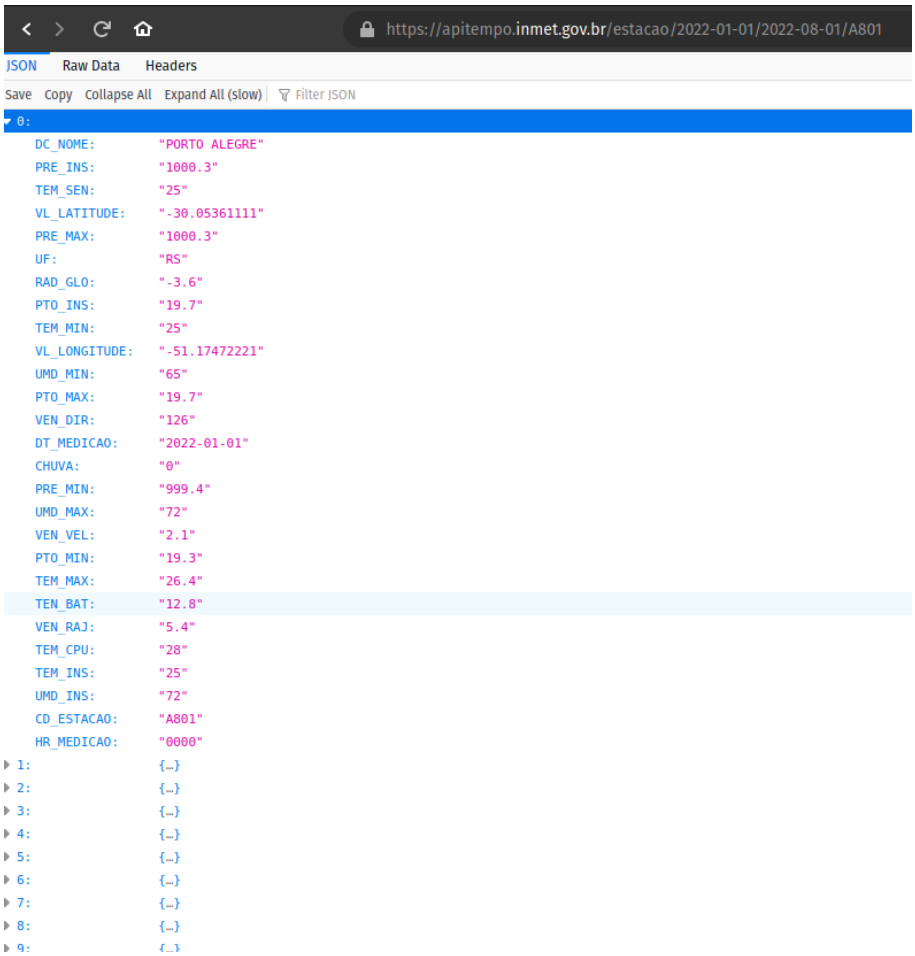
- https://github.com/yanhaeffner/tcc

APÊNDICE B - IMAGENS

B.1 API INMET

O retorno da API do INMET para a solicitação de dados da estação meteorológica de Porto Alegre entre janeiro e agosto de 2022 é apresentado na Figura 28. A resposta retorna uma lista de elementos (cada elemento representado pelo seu índice da lista) JSON com valores textuais lidos da estação meteorológica para cada hora das datas solicitadas.

Figura 28: *Resposta de solicitação de dados INMET da estação de Porto Alegre entre janeiro e agosto de 2022.*

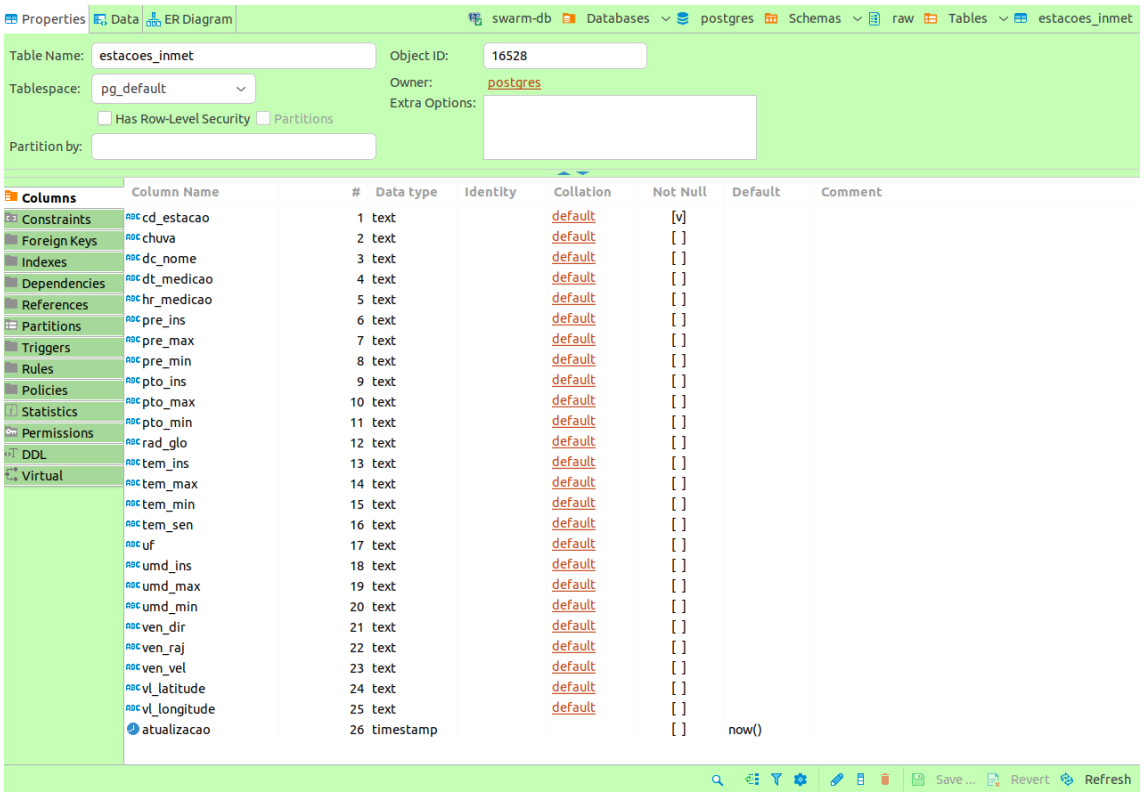


Fonte: O autor.

B.2 Carregamento de Dados

A estrutura da tabela de dados meteorológicos da área *raw* é apresentada na Figura 29, onde é possível perceber que todas as informações coletadas dos arquivos disponíveis são escritas como elementos textuais puros, evitando assim conflitos durante o processo de carregamento.

Figura 29: *Tabela RAW para dados de estações meteorológicas (INMET) em banco PostgreSQL.*



Fonte: O autor.

A Figura 30 apresenta a relação de dados de estações meteorológicas de *raw* para a área *stg* no *Data Warehouse*, sendo em sua maioria uma conversão direta entre formato textual para formato numérico.

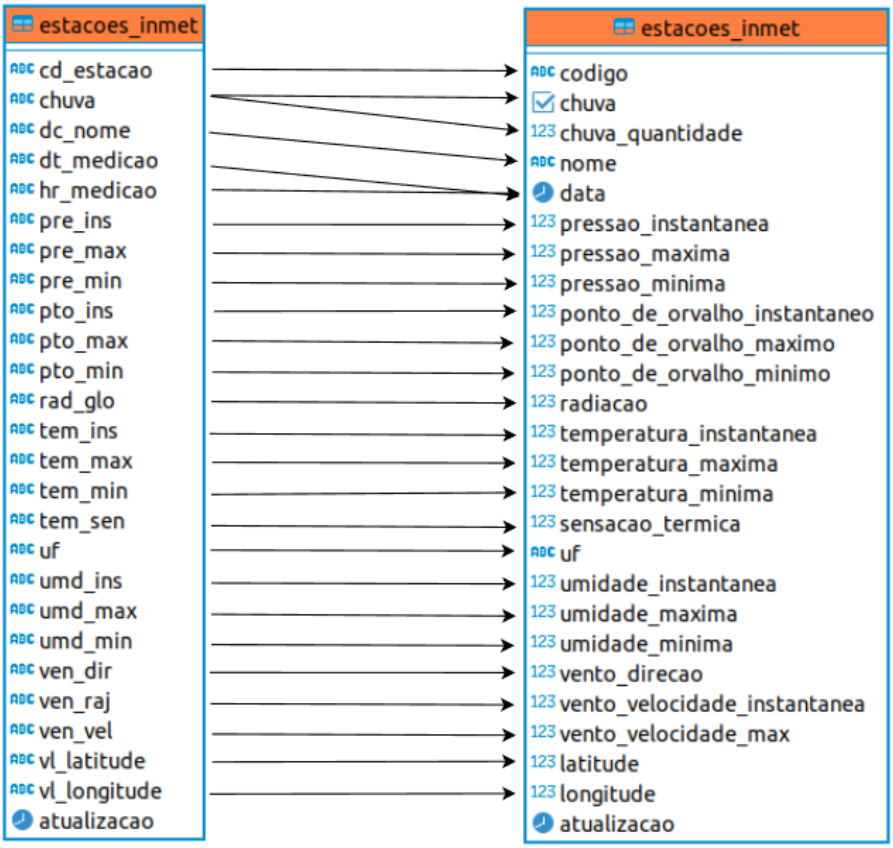
B.3 Tratamento de Dados

A Figura 31 apresenta a relação dos eventos de sensores e atuadores das estações de irrigação de Porto Alegre (V801) com as distâncias das estações meteorológicas disponíveis em *dim_geo*, apresentando a estação de Porto Alegre como a mais próxima, dado o valor da coluna *distancia* em quilômetros, conforme esperado pela regra de negócio estabelecida.

B.4 DAGs

A Figura 32 apresenta a visão de diagrama do DAG de processamento de dados INMET contendo estações adicionais utilizadas para os testes de escalonamento. A quantidade de

Figura 30: *Relação entre as tabelas raw.estacoes_inmet e stg.estacoes_inmet.*



Fonte: O autor.

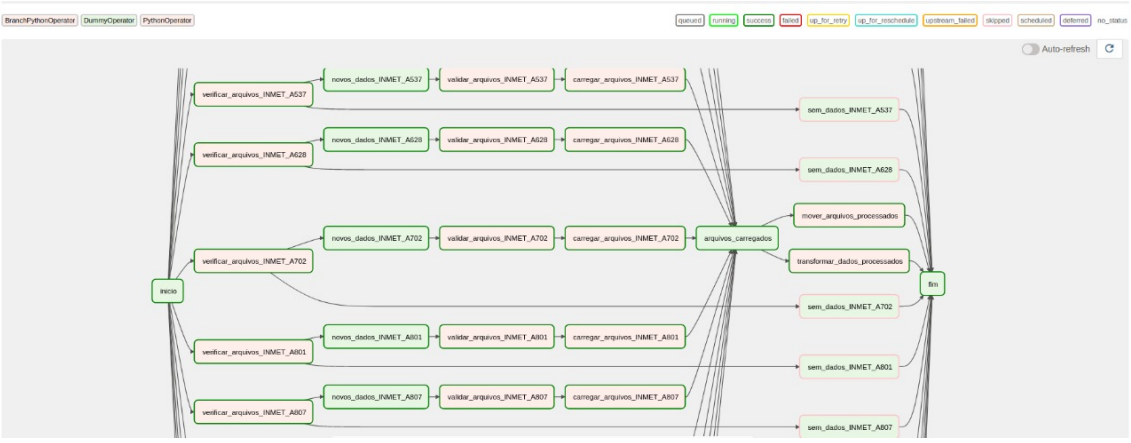
setas saindo da tarefa de início indica a presença de inúmeras tarefas adicionais criadas dinamicamente para o processamento individual das estações.

Figura 31: *Distâncias das estações meteorológicas dos eventos da estação de irrigação de Porto Alegre.*

	cliente	tipo	nome	nome_estacao	uf	distancia	rn
1	V801	atuador	bomba_hidraulica	PORTO ALEGRE	RS	4.7266819716	1
2	V801	atuador	bomba_hidraulica	CANELA	RS	82.7397785282	2
3	V801	atuador	bomba_hidraulica	CAMAQUÃ	RS	104.8861862775	3
4	V801	atuador	bomba_hidraulica	PASSO FUNDO	RS	231.6499918112	4
5	V801	atuador	bomba_hidraulica	CHAPECO	SC	356.0507937324	5
6	V801	atuador	bomba_hidraulica	CURITIBA	PR	546.2845629263	6
7	V801	atuador	bomba_hidraulica	ANGRA DOS REIS	RJ	1,043.9855303577	7
8	V801	atuador	bomba_hidraulica	CAMPO GRANDE	MS	1,123.5197007128	8
9	V801	atuador	bomba_hidraulica	BELO HORIZONTE (PAMPULHA)	MG	1,344.7342970052	9
10	V801	atuador	bomba_hidraulica	DIAMANTINA	MG	1,520.837331688	10
11	V801	atuador	bomba_hidraulica	BRASILIA	DF	1,620.5439381022	11
12	V801	atuador	bomba_hidraulica	CAMPO VERDE	MT	1,663.0171953552	12
13	V801	atuador	bomba_hidraulica	PALMAS	TO	2,229.1588011437	13
14	V801	atuador	bomba_hidraulica	CAXIAS	MA	2,925.6393301574	14
15	V801	sensor	luminosidade	PORTO ALEGRE	RS	4.7266819716	1
16	V801	sensor	luminosidade	CANELA	RS	82.7397785282	2
17	V801	sensor	luminosidade	CAMAQUÃ	RS	104.8861862775	3
18	V801	sensor	luminosidade	PASSO FUNDO	RS	231.6499918112	4
19	V801	sensor	luminosidade	CHAPECO	SC	356.0507937324	5
20	V801	sensor	luminosidade	CURITIBA	PR	546.2845629263	6
21	V801	sensor	luminosidade	ANGRA DOS REIS	RJ	1,043.9855303577	7
22	V801	sensor	luminosidade	CAMPO GRANDE	MS	1,123.5197007128	8
23	V801	sensor	luminosidade	BELO HORIZONTE (PAMPULHA)	MG	1,344.7342970052	9
24	V801	sensor	luminosidade	DIAMANTINA	MG	1,520.837331688	10
25	V801	sensor	luminosidade	BRASILIA	DF	1,620.5439381022	11
26	V801	sensor	luminosidade	CAMPO VERDE	MT	1,663.0171953552	12
27	V801	sensor	luminosidade	PALMAS	TO	2,229.1588011437	13
28	V801	sensor	luminosidade	CAXIAS	MA	2,925.6393301574	14
29	V801	sensor	umidade_solo	PORTO ALEGRE	RS	4.7266819716	1
30	V801	sensor	umidade_solo	CANELA	RS	82.7397785282	2
31	V801	sensor	umidade_solo	CAMAQUÃ	RS	104.8861862775	3
32	V801	sensor	umidade_solo	PASSO FUNDO	RS	231.6499918112	4
33	V801	sensor	umidade_solo	CHAPECO	SC	356.0507937324	5

Fonte: O autor.

Figura 32: *Captura de tela de gráfico do DAG de processamento de estações INMET expandidas*



Fonte: O autor.