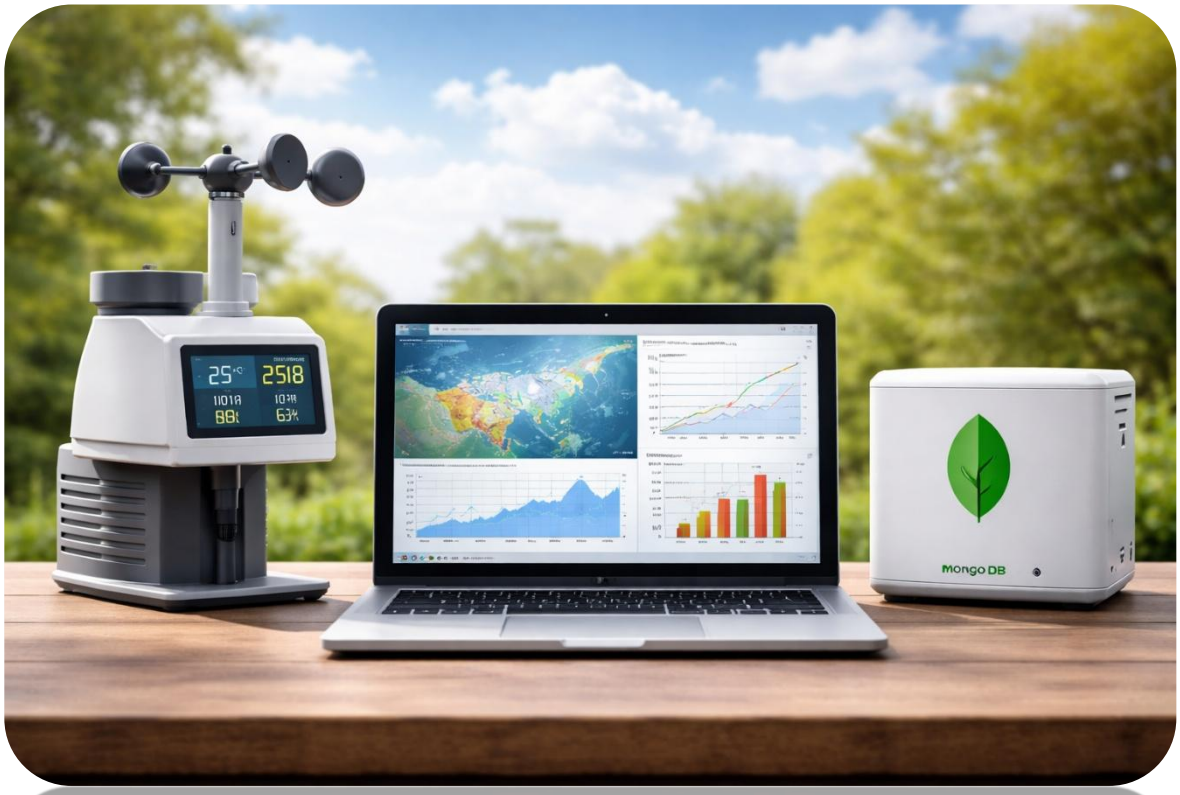
Politécnico de Tomar
Escola Superior de Tecnologia de Abrantes



Predição Meteorológica

https://meteoabrantes.lcerejo.com

Politécnico de Tomar
Escola Superior de Tecnologia de Abrantes

Escola Superior de Tecnologia de Abrantes

Meteo Abrantes:

Real vs Previsto

CTeSP Informática 2º Ano, 1º Semestre

Luís Cerejo, nº 82674

Unidade Curricular: Projeto Integrado

Docente: Paulo Rêgo

Abrantes, 29 de Dezembro de 2025

AGRADECIMENTOS

Gostaria de expressar o meu sincero agradecimento ao Professor Paulo Rêgo pelo acompanhamento, disponibilidade e orientação ao longo do desenvolvimento deste projeto. As sugestões técnicas e conceptuais apresentadas, nomeadamente a introdução e recomendação de aplicações e ferramentas que vieram enriquecer significativamente a solução desenvolvida, foram determinantes para a evolução do projeto. Sem essas indicações, muitas das funcionalidades e abordagens adotadas não teriam sido exploradas. Este contributo revelou-se essencial para o aprofundamento dos meus conhecimentos, tanto a nível técnico como metodológico, resultando num trabalho mais consistente, estruturado e com maior valor tanto académico como prático.

RESUMO

A ideia subjacente ao projeto AbrantesMeteo consistiu no desenvolvimento de uma plataforma integrada para a recolha, previsão e a análise de dados meteorológicos na cidade de Abrantes. Combinou-se dados históricos importados de uma base de dados externa, medições em tempo real e utilização de modelos preditivos (prophet), com visualização centralizada em dashboards produzidos em Power BI. O objetivo principal é demonstrar, em contexto puramente académico, a aplicação prática de tecnologias de IoT, bases de dados NoSQL, modelação preditiva e Business Intelligence, assegurando a atualização automática dos dados, bem como a sua consistência ao longo do tempo.

Na fase inicial, o sistema recolhe os dados meteorológicos históricos e diários através da API Open-Meteo, armazenando-os numa base de dados MongoDB em ambiente Docker (modo local). Estes dados incluem os registos de temperatura mínima, máxima e média, bem como a precipitação diária registada, constituindo a base histórica do projeto desde 1 de janeiro de 2023. A atualização ocorre diariamente de forma automática, garantindo a continuidade temporal da informação.

Foi sobre esta base de dados históricos que se implementou um módulo de previsão meteorológica em Python, recorrendo ao modelo Prophet, que gera previsões diárias para um horizonte temporal definido para 365 dias. As previsões são armazenadas separadamente dos dados reais, permitindo análises comparativas entre valores observados e valores previstos, bem como a avaliação de tendências futuras.

O projeto foi posteriormente expandido com a integração de um Raspberry Pi 5 equipado com um sensor BME280, permitindo a recolha de dados físicos reais de temperatura, humidade e pressão atmosférica. Estas medições locais acrescentam uma componente experimental ao sistema e são igualmente armazenadas localmente em MongoDB, possibilitando visualizações em “tempo real” no Power BI.

Para garantir a atualização automática dos dashboards no Power BI Service, os dados são sincronizados para o MongoDB Atlas e expostos através de Atlas SQL (ODBC), solução compatível com o On-premises Data Gateway. Desta forma, todo o ecossistema utiliza uma arquitetura consistente e robusta, evitando limitações de conectividade direta ao MongoDB.

Em resumo, o projeto AbrantesMeteo apresenta uma solução completa e escalável, que integra recolha automática de dados, previsão estatística e visualização analítica, evidenciando técnicas de engenharia de dados, automação e análise preditiva, com uma abordagem alinhada com práticas profissionais.

Índice

1. Introdução 7
2. Objetivos e critérios de sucesso 7
3. Ambiente de execução e tecnologias 8
3.1. Infraestrutura 8
3.2. Arquitetura tecnológica 8
4. Arquitetura lógica e fluxo de dados 9
5. Estrutura do projeto no servidor (docker-ct) 9
5.1. Função dos diretórios e ficheiros 10
6. Implementação detalhada 12
6.1. Preparação do diretório e ficheiros base 12
6.2. Artefactos de configuração e deployment (Docker) 13
6.2.1. Orquestração de serviços (docker-compose.yml) 14
6.2.2. Parametrização do ambiente (.env) 14
6.2.3. Dependências Python (requirements.txt) 14
6.2.4. Inicialização da base de dados (01-setup.js) 15
6.2.5. Construção da imagem da aplicação (Dockerfile) 15
6.2.6. Pipeline aplicacional (app.py) 15
6.3. Execução e validação do MongoDB local 16
6.4. Sincronização para MongoDB Atlas 16
6.4.1. Dificuldades típicas e mitigação 18
6.5. Exposição ao Power BI via Data Federation e Atlas SQL (ODBC) 18
6.5.1. Mapeamento de coleções na Federated Database 19
6.5.2. Geração do schema SQL (resolução do erro “no visible columns”) 19
6.5.3. Configuração do driver ODBC (DSN) 20
6.6. Modelação no Power BI 21
6.6.1. Tabela calendário 21
6.6.2. Relações (Model View) 22
6.6.3. forecast_fixo: baseline de previsão 22
6.6.4. Medidas DAX para comparação (exemplo: Tmax, Tmin e Precip) 24
6.7. Integração de Raspberry Pi 5 + sensor BME280 25
6.7.1. Arquitetura e fluxo de dados 25
6.7.2. Preparação do hardware e I2C no Raspberry Pi 5 26
6.7.3. Projeto Python no Raspberry (venv, dependências e ficheiro .env) 26
6.7.4. Script de recolha e envio para o Atlas 27
6.7.5. Automatização no Raspberry (cron: 5 leituras diárias) 27

6.7.6. Monitorização local 28
6.7.7. Evidência de integração (Raspberry Pi → Atlas) 28
6.7.8. Atlas: permissões, coleção e schema para Power BI (Atlas SQL Interface) 29
6.7.9. Power BI Desktop: importar a tabela e criar o scroller 30
7. Dificuldades & Resolução 31
8. Resultados finais obtidos 31
9. Melhorias futuras e visão de evolução 32
10. Conclusão 34
11. Glossário 35
12. Referências online 37
13. Anexos 38
No servidor (docker) 38
Anexo A - docker-compose.yml 38
Anexo B - .env 40
Anexo C - requirements.txt 42
Anexo D - mongo-init/01-setup.js 43
Anexo E - app/Dockerfile 45
Anexo F - app.py (pipeline completo) 46
No Raspberry PI 53
Anexo G - Raspberry Pi: monitor_local.sh 53
Anexo H - Raspberry Pi: read_local.py 54
Anexo I - Raspberry Pi: run_once.sh 55
Anexo J - Raspberry Pi: sensor_to_atlas.py 56
Anexo L - Organograma Inicial após 1ª revisão 58
Anexo M - Organograma projeto final 59

1. Introdução

O projeto AbrantesMeteo é apresentado como um caso de estudo que visa demonstrar uma arquitetura pensada e implementada para a recolha, tratamento e disponibilização de dados, aplicável a qualquer domínio (operações, produção, vendas, etc.) - e não somente o tratamento de dados meteorológicos. O objetivo principal é validação, em contexto académico, de uma cadeia de dados de ponta a ponta: ingestão de fontes heterogéneas, normalização e validação, armazenamento estruturado, criação de métricas e modelos (quando aplicável) e, por fim, consumo em dashboards para análise e decisão.

A componente meteorológica pretende recriar um cenário realista para testar requisitos típicos de um pipeline de informação: automatização, repetibilidade, auditável (logs e rastreio de execuções), separação de responsabilidades (módulos e serviços independentes) e qualidade dos dados (consistência temporal, controlo de duplicados, tratamento de falhas). O que se pretende, mais do que "fazer gráficos e tabelas informativos", é evidenciar um processo operacional robusto e funcional que pode ser reutilizado e escalado para outros contextos.

Ao longo do relatório são descritos os objetivos e requisitos, a arquitetura e tecnologias adotadas, a implementação passo a passo (incluindo organização de ficheiros/serviços), as principais dificuldades técnicas e respetivas correções, os resultados obtidos no Power BI e um conjunto de melhorias futuras orientadas para maior resiliência, escalabilidade e manutenção da solução.

---

2. Objetivos e critérios de sucesso

- Recolher histórico meteorológico (Open-Meteo archive) e armazenar em MongoDB local uma vez por dia.
- Gerar previsões diárias (365 dias) com Prophet, normalizando o output em formato "flat" para análise tabular.
- Sincronizar automaticamente para MongoDB Atlas, assegurando autenticação, segregação de utilizadores e permissões mínimas.
- Expor os dados ao Power BI via Atlas Data Federation/Atlas SQL (ODBC), evitando dependência de drivers "MongoDB nativos" no Power BI.
- Construir um modelo de dados no Power BI com tabela de calendário, medidas DAX e visualizações para comparação real vs. previsto.

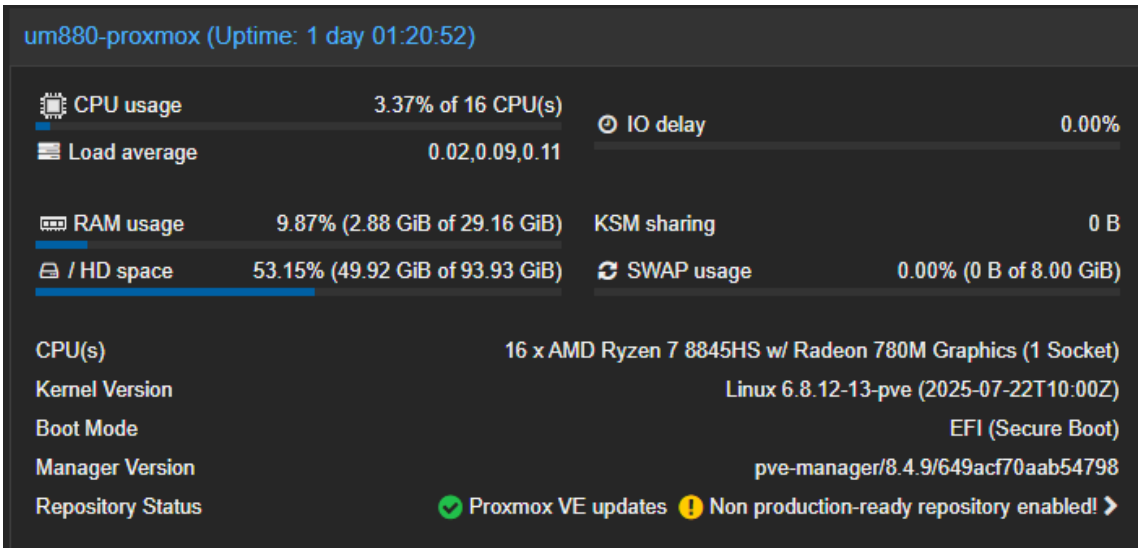
- Garantir atualização diária e consistência do baseline de previsão (forecast_fixo) para comparar ‘o que estava previsto’ com o que aconteceu.

---

3. Ambiente de execução e tecnologias

3.1. Infraestrutura

- Servidor: MiniPC UM880 (host Proxmox).
- Execução: contentor LXC dedicado (“docker-ct”) a correr Docker Engine e Docker Compose.
- Cliente analítico: PC Windows (Power BI Desktop) e, opcionalmente, Power BI Service com On-premises Data Gateway (Personal Mode).



3.2. Arquitetura tecnológica

- MongoDB 7 (container) para armazenamento local e testes/validação.
- MongoDB Atlas (cluster) para persistência cloud e consumo analítico.
- Atlas Data Federation + Atlas SQL para exposição SQL/ODBC.
- Python 3.11 para ingestão, normalização e previsão.
- Prophet (cmdstanpy) para forecasting; requests/pymongo/pandas para integração e transformação.
- Power BI Desktop para modelação, DAX e dashboards; ODBC para ligação ao Atlas SQL.

4. Arquitetura lógica e fluxo de dados

A solução proposta foi projetada como um pipeline modular. A ingestão e a previsão correm no servidor, junto ao armazenamento local, garantindo execução diária e independência do posto de trabalho. A cloud (Atlas) é usada para centralizar e disponibilizar os dados, permitindo que o Power BI aceda de forma consistente (ODBC) e que o refresh seja operacionalizado via Gateway quando necessário.

Fonte (Open-Meteo)
→ Serviço Python (ingestão + Prophet)
→ MongoDB local (abrantes_meteo)
→ Sync (export/import)
→ MongoDB Atlas
→ Data Federation/Atlas SQL
→ ODBC (DSN)
→ Power BI (Modelo + Medidas + Dashboards)

5. Estrutura do projeto no servidor (docker-ct)

A implementação foi organizada numa pasta dedicada dentro do RAID/volume do servidor, assegurando persistência dos dados (mongo-data) e separação entre código, logs e segredos. A estrutura abaixo corresponde à pasta /mnt/storage/abrantes_meteo.

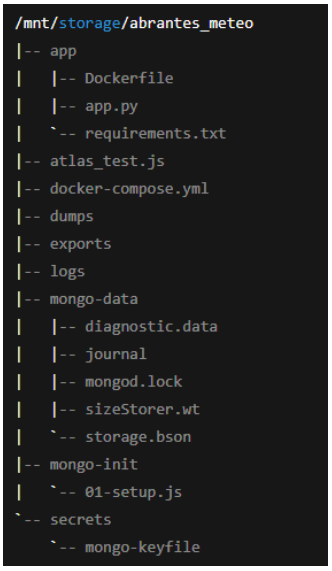


Figura 1 - Estrutura de pastas do projeto no servidor (diretório /mnt/storage/abrantes_meteo).

5.1. Função dos diretórios e ficheiros

a) *docker-compose.yml*

Define os serviços Docker do projeto:

- mongodb (MongoDB 7) com --replSet rs0, --auth e --keyFile (requisito técnico para auth + replicaset).
- mongo_setup (container temporário) que executa mongo-*init/01-setup.js* após o Mongo estar saudável.
- app (Python) que faz backfill, previsão, sync e scheduler.

```yaml
services:
  mongodb:
    image: mongo:7
    container_name: mongodb
    restart: unless-stopped
    ports:
      - "27017:27017"
    environment:
      MONGO_INITDB_ROOT_USERNAME: root
      MONGO_INITDB_ROOT_PASSWORD: "lcerejo"
      MONGO_INITDB_DATABASE: projeto
    volumes:
      - ./mongo-data:/data/db
      - ./secrets/mongo-keyfile:/etc/mongo-keyfile:ro
    command: ["--bind_ip_all","--replSet","rs0","--keyFile","/etc/mongo-keyfile","--auth"]
    healthcheck:
      test: ["CMD-SHELL", "mongosh --quiet --eval 'db.adminCommand({ping:1})' || exit 1"]
      interval: 10s
      timeout: 5s
      retries: 12

  meteo_cron:
    image: python:3.11-slim
    container_name: meteo_cron
    restart: unless-stopped
    depends_on:
      mongodb:
        condition: service_healthy
    working_dir: /app
    volumes:
      - ./scripts:/app
    environment:
      TZ: "Europe/Lisbon"
      LAT: "39.46"
      LON: "-8.2"
      RUN_HOUR_LOCAL: "10"
      LOCAL_URI: "mongodb://app:lcerejo@mongodb:27017/projeto?replicaSet=rs0&authSource=projeto&authMechanism=SCRAM-SHA-256"
    command:
      - /bin/sh
      - -c
      - |
        pip install --no-cache-dir requests pymongo &&
        python /app/fetch_meteo.py --daemon ;
        tail -f /dev/null
```

Figura 2 - Ficheiro docker-compose.yml do projeto (serviços e volumes definidos).

b) *secrets/mongo-keyfile*

Keyfile usado pelo MongoDB para permitir replicaset com autenticação ativa. É gerado localmente com permissões restritas (chmod 400) e montado read-only no container.

c) *mongo-init/01-setup.js*

Script idempotente que:

- inicializa replicaset (rs0) se necessário,
- cria a base abrantes_meteo e o utilizador de aplicação (app) com readWrite,
- cria índices essenciais (ex.: meteo_historico.date único; forecast indexado por run_id + date e por created_at).

d) app/Dockerfile e app/requirements.txt

Constroem a imagem da aplicação:

- Base python:3.11-slim,
- Instala dependências (requests, pymongo, pandas, prophet, cmdstanpy),
- Define app.py como entrypoint.

e) app/app.py

É o núcleo do sistema e implementa:

- **Recolha histórica** via Open-Meteo Archive (por blocos, para estabilidade),
- **Upsert** diário (não duplica dias e mantém consistência),
- **Treino e previsão** com Prophet para as variáveis meteorológicas,
- Escrita do forecast com run_id diário e created_at,
- **Sync para Atlas** (histórico por janela lookback + forecast do run_id atual),
- **Scheduler**: corre uma vez no arranque e depois diariamente na hora definida.

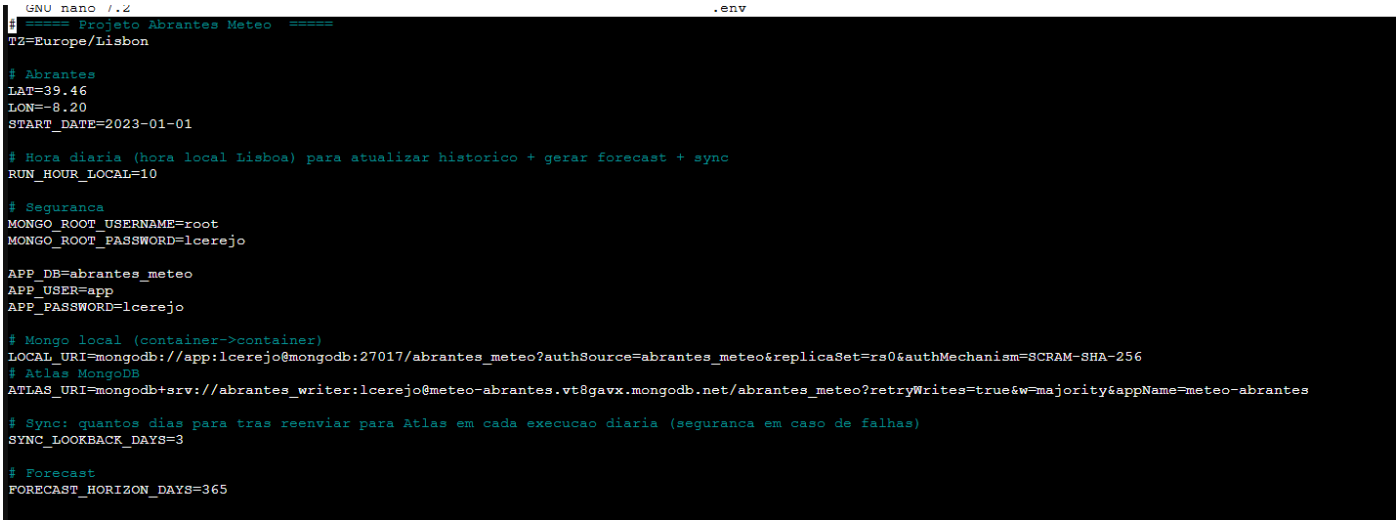


Figura 3 - Variáveis de ambiente no ficheiro .env (fuso horário e hora de execução do scheduler).

f) mongo-data/ (persistência da base de dados local)

Esta pasta é um bind mount para /data/db do container MongoDB, garantindo persistência após reinícios.

Os ficheiros internos (ex.: journal/, sizeStorer.wt, mongod.lock, diagnostic.data, storage.bson) são artefactos normais do motor WiredTiger, e a sua gestão é feita pelo MongoDB.

Esta decisão assegura:

- retenção do histórico e forecasts locais,
- recuperação rápida após falhas de energia,

- independência do Atlas para o funcionamento base (Atlas é consumo/sync).

logs/, exports/, dumps/

Pastas operacionais:

- logs/: evidência operacional (logs de execução e troubleshooting).
- exports/: suporte a exportações para ficheiros (quando necessário para auditoria/partilha).
- dumps/: suporte a backups com mongodump/mongorestore.
  (Em contexto académico, são úteis como "instrumentação" e preparação para operação em produção.)

g) atlas_test.js

Script de teste para validar conectividade e queries no Atlas via mongosh, garantindo que:

- credenciais e URI estão corretos,
- coleções existem e estão acessíveis,
- o ambiente está operacional para o Power BI.

6. Implementação detalhada

Esta secção pretende descrever o processo de implementação de uma forma sequencial. Sempre que foram necessárias decisões de configuração, as mesmas são justificadas tecnicamente, indicando também os problemas encontrados e a respetiva resolução.

6.1. Preparação do diretório e ficheiros base

O ponto de partida foi a criação da pasta de trabalho no servidor e a definição da estrutura mínima (app, mongo-init, secrets, volumes). O objetivo é permitir que todo o ciclo de vida (deploy, restart, upgrades) seja controlado por Docker Compose.

```
# (no docker-ct)
mkdir -p /mnt/storage/abrantes_meteo/{app,mongo-init,secrets,mongo-
data,logs,exports,dumps}
cd /mnt/storage/abrantes_meteo
```

```
env (exemplo)
TZ=Europe/Lisbon
LAT=39.46
```

```
LON=-8.20
START_DATE=2023-01-01
RUN_HOUR=10

# Mongo local
MONGO_DB=abrantes_meteo
MONGO_USER=app
MONGO_PASS=<LOCAL_PASSWORD>
LOCAL_URI=mongodb://app:<LOCAL_PASSWORD>@mongodb:27017/abrantes_meteo?authSource=abrantes_
meteo

# Mongo Atlas (writer/reader)
ATLAS_URI=mongodb+srv://abrantes_writer:<ATLAS_PASSWORD>@meteo-
abrantes.<cluster>.mongodb.net/abrantes_meteo?retryWrites=true&w=majority&appName=meteo-
abrantes
```

6.2. Artefactos de configuração e deployment (Docker)

Nesta fase, o sistema é encapsulado e executado em ambiente **Docker** no host/CT, garantindo portabilidade, reprodutibilidade e reinício automático após falhas (ex.: corte de energia). Os ficheiros críticos de configuração e construção foram isolados em artefactos próprios. O detalhe integral de cada ficheiro encontra-se nos anexos, evitando duplicação de código no corpo do relatório.

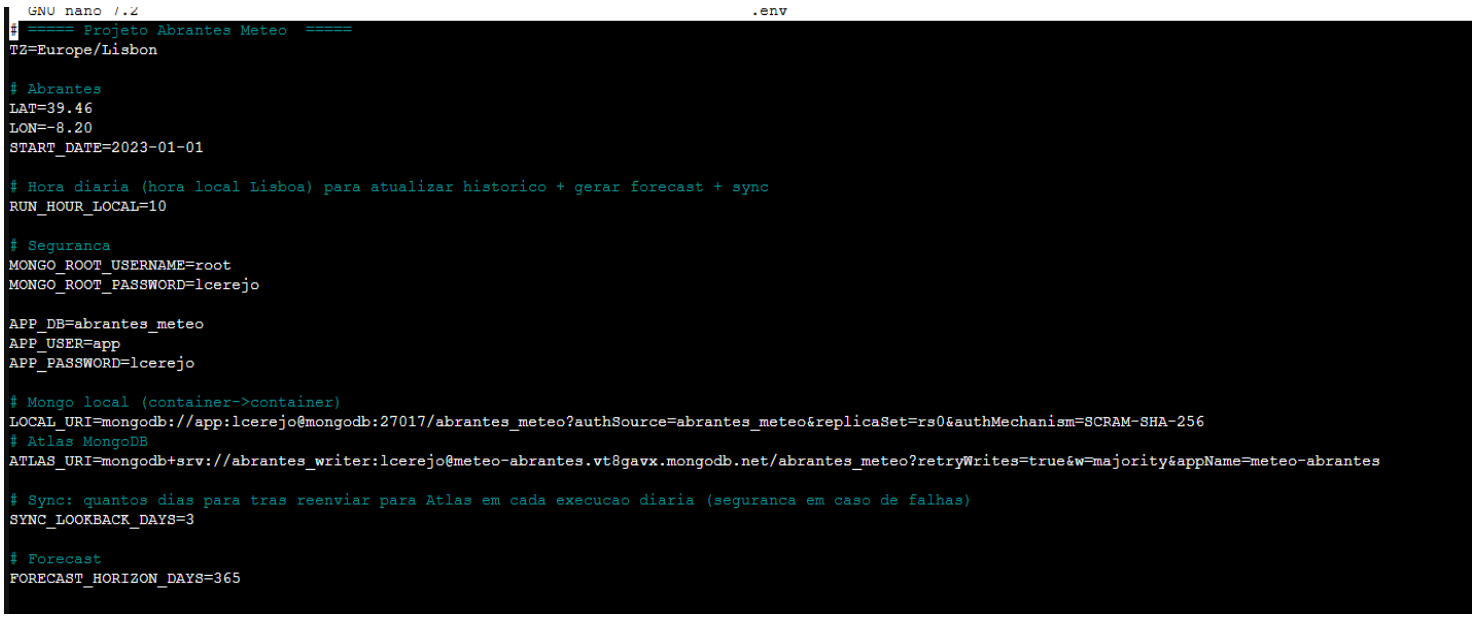


Figura 4 - Variáveis de ambiente no ficheiro .env (fuso horário e hora de execução do scheduler).

6.2.1. Orquestração de serviços (docker-compose.yml)

O ficheiro docker-compose.yml define a arquitetura operacional do AbrantesMeteo em contentores, incluindo:

- serviço de base de dados (MongoDB) com persistência em volume/diretório;
- serviço de inicialização/"bootstrap" (ex.: utilizadores/roles/replica set, quando aplicável);
- serviço da aplicação Python (pipeline diário: ingestão → persistência → previsão → sincronização);
- políticas de reinício para resiliência (reinício automático após reboot/falha).

Ver: **Anexo A - docker-compose.yml.**

6.2.2. Parametrização do ambiente (.env)

O ficheiro .env centraliza variáveis de execução, permitindo ajustar o comportamento do sistema sem alterações ao código, nomeadamente:

- localização e fuso horário (LAT, LON, TZ);
- data de início do histórico (START_DATE);
- ligações a MongoDB local e MongoDB Atlas (LOCAL_URI, ATLAS_URI);
- base de dados e parâmetros do ciclo diário (APP_DB, RUN_HOUR_LOCAL);
- horizonte de previsão e janela de sincronização (FORECAST_HORIZON_DAYS, SYNC_LOOKBACK_DAYS).

Ver: **Anexo B -.env**

6.2.3. Dependências Python (requirements.txt)

O requirements.txt fixa o conjunto de bibliotecas necessárias para:

- acesso HTTP à API meteorológica;
- manipulação de dados (estruturas tabulares/séries temporais);
- acesso a MongoDB;
- execução do modelo preditivo (Prophet).

Ver: **Anexo C — requirements.txt.**

6.2.4. Inicialização da base de dados (01-setup.js)

O script 01-setup.js suporta a preparação do MongoDB para operação estável, tipicamente:

- criação/validação de utilizadores e permissões (quando aplicável);
- criação/validação de coleções e índices base;
- execução idempotente (pode correr mais do que uma vez sem alterar incorretamente o estado).

Ver: **Anexo D — 01-setup.js.**

6.2.5. Construção da imagem da aplicação (Dockerfile)

O Dockerfile define como é construída a imagem do serviço aplicacional, assegurando:

- base runtime consistente (Python);
- instalação de dependências conforme requirements.txt;
- cópia controlada do código;
- comando de arranque explícito (execução do app.py no contentor).

Ver: **Anexo E — Dockerfile.**

6.2.6. Pipeline aplicacional (app.py)

O ficheiro app.py implementa o ciclo diário do AbrantesMeteo, assegurando a execução completa do pipeline de dados desde a ingestão até à disponibilização para análise. Em termos funcionais, o ficheiro inclui:

- Ingestão de dados históricos/diários a partir da API Open-Meteo, com normalização de campos e datas;
- Persistência no MongoDB local com escrita idempotente (atualização sem duplicação), garantindo continuidade temporal;
- Geração de previsões (Prophet) para o horizonte configurado, com identificação por execução diária através de run_id;
- Sincronização para MongoDB Atlas, enviando histórico recente (janela de segurança) e a previsão do dia para consumo externo (Power BI);
- Agendamento automático, que corre uma primeira vez no arranque e volta a executar diariamente à hora definida.

Ver: **Anexo F — app.py.**

6.3. Execução e validação do MongoDB local

Após a configuração dos ficheiros e serviços, o deployment é efetuado com Docker Compose. O objetivo desta etapa é garantir que o stack fica operacional e que o pipeline executa corretamente, validando:

- disponibilidade e estabilidade do serviço MongoDB (incluindo persistência);
- execução do carregamento inicial do histórico (backfill), quando aplicável;
- geração da previsão diária (forecast) para o horizonte definido;
- escrita consistente nas coleções principais do projeto.

A validação foi realizada através do cliente mongosh, confirmando contagens de documentos e a existência de registos recentes nas coleções relevantes. Esta verificação é importante para distinguir falhas de ingestão/persistência de problemas exclusivamente ligados à camada de visualização (Power BI).

```
docker exec -it abrantes_meteo_mongodb mongosh -u root -p "<ROOT_PASSWORD>" --
authenticationDatabase admin --quiet --eval '
db=db.getSiblingDB("abrantes_meteo");
print("HIST=", db.meteo_historico.countDocuments());
print("FORE=", db.meteo_forecast_flat.countDocuments());
printjson(db.meteo_historico.find().sort({date:-1}).limit(1).toArray()[0]);
printjson(db.meteo_forecast_flat.find().sort({created_at:-
1}).limit(1).toArray()[0]);'
```

6.4. Sincronização para MongoDB Atlas

Após validação do armazenamento local, foi configurado um cluster MongoDB Atlas para alojar a base de dados em cloud e suportar o consumo analítico (Power BI via Atlas/ODBC). A configuração incluiu a criação do projeto e do cluster, bem como a definição das regras de acesso à infraestrutura (IP Access List) e das credenciais de base de dados.

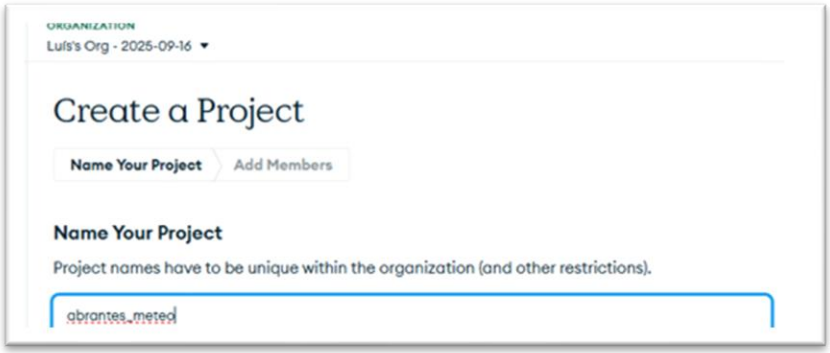


Figura 5 - Criação do projeto no MongoDB Atlas (console Atlas).

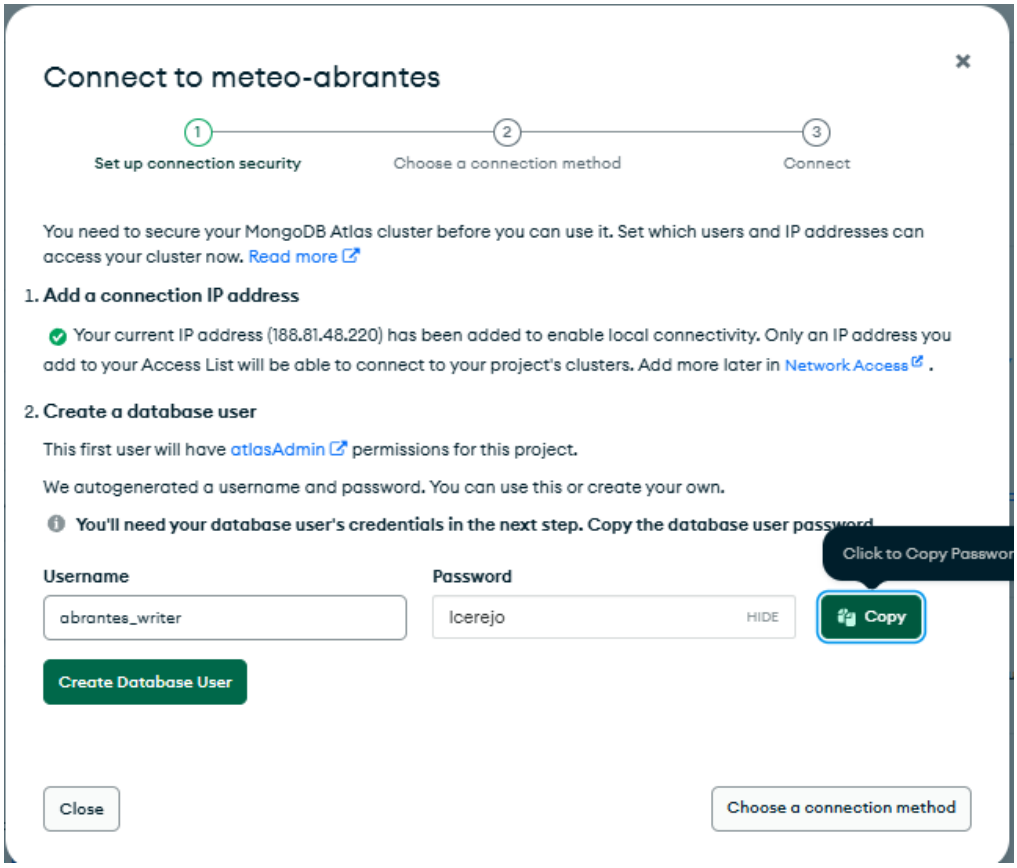


Figura 6 - Configuração de acesso ao cluster no Atlas (IP Access List e utilizador de base de dados).

Para separar responsabilidades e aplicar o princípio do menor privilégio, foram definidos perfis de acesso distintos:

- **abrantes_writer**: permissões de escrita na base abrantes_meteo, utilizado pelos processos de ingestão e sincronização;
- **abrantes_reader**: permissões de leitura na base abrantes_meteo, utilizado para consumo analítico (Atlas SQL/ODBC e Power BI).

Após integração do Raspberry Pi, foram adicionadas credenciais adicionais específicas para esse fluxo, mantendo a lógica de segregação por função.

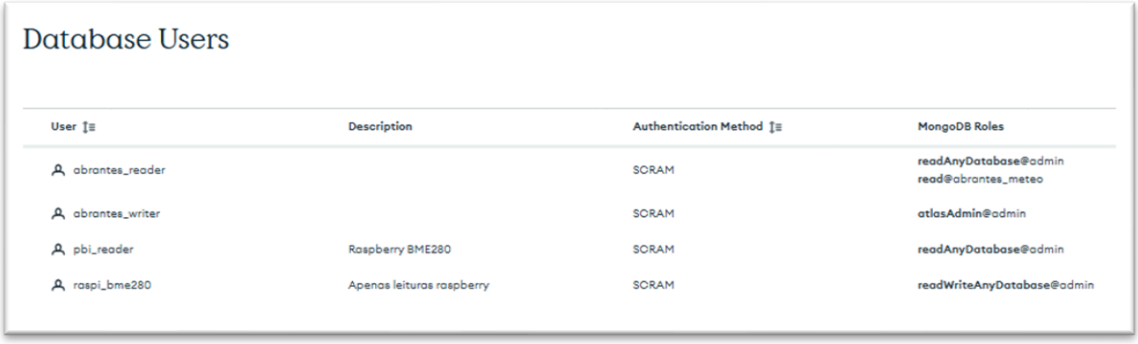


Figura 7 - Gestão de utilizadores no Atlas (Database Users) para acesso controlado ao cluster.

Concluída a configuração, a ligação ao Atlas foi parametrizada através da variável **ATLAS_URI** no ficheiro .env e a conectividade foi validada, confirmando a disponibilidade dos dados sincronizados. Ver:

- Anexo B — .env (parametrização do ATLAS_URI)
- Anexo F — app.py (lógica de sincronização para Atlas)

6.4.1. Dificuldades típicas e mitigação

Durante a configuração do Atlas e dos testes de ligação, surgiram situações típicas relacionadas com:

- **autenticação** (credenciais incorretas ou permissões insuficientes);
- **variáveis de ambiente** não carregadas corretamente antes da execução;
- **restrições de rede** (IP não autorizado na Access List);
- **URI/forma de ligação** (uso incorreto de host em vez de URI do Atlas).

A resolução passou por confirmar, de forma sistemática, permissões do utilizador, carregamento correto de variáveis, autorização de IP e utilização da URI recomendada pelo Atlas.

6.5. Exposição ao Power BI via Data Federation e Atlas SQL (ODBC)

O Power BI não consome MongoDB de forma nativa com a mesma maturidade com que consome fontes relacionais. Por esse motivo, foi adotada a abordagem Atlas Data Federation + Atlas SQL Interface, permitindo expor as coleções do cluster Atlas através de um endpoint SQL e consumi-las no Power BI via ODBC.

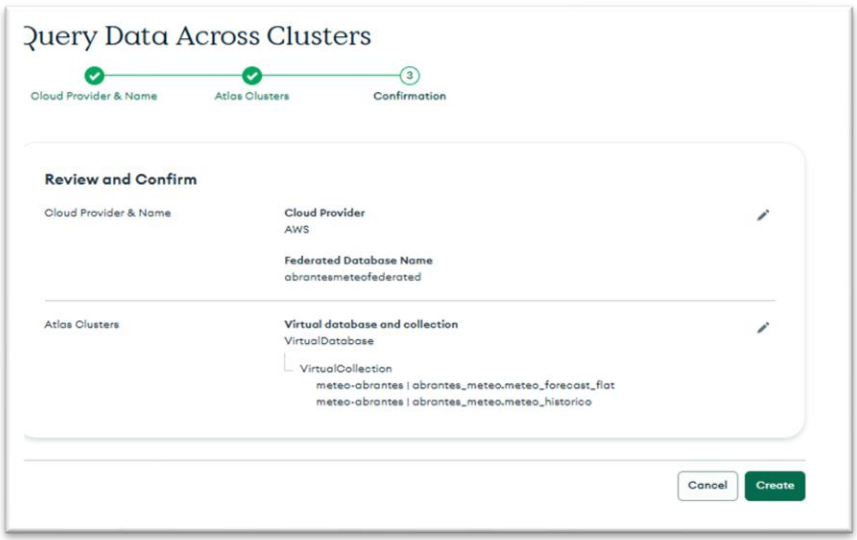


Figura 8 - Criação da Federated Database Instance (Atlas Data Federation) para expor dados do cluster.

6.5.1. Mapeamento de coleções na Federated Database

Na configuração da Federated Database Instance, foram mapeadas as coleções do cluster Atlas necessárias ao projeto, nomeadamente:

- meteo_historico
- meteo_forecast_flat

A Virtual Database foi normalizada com um nome consistente para evitar ambiguidades e problemas de integração.

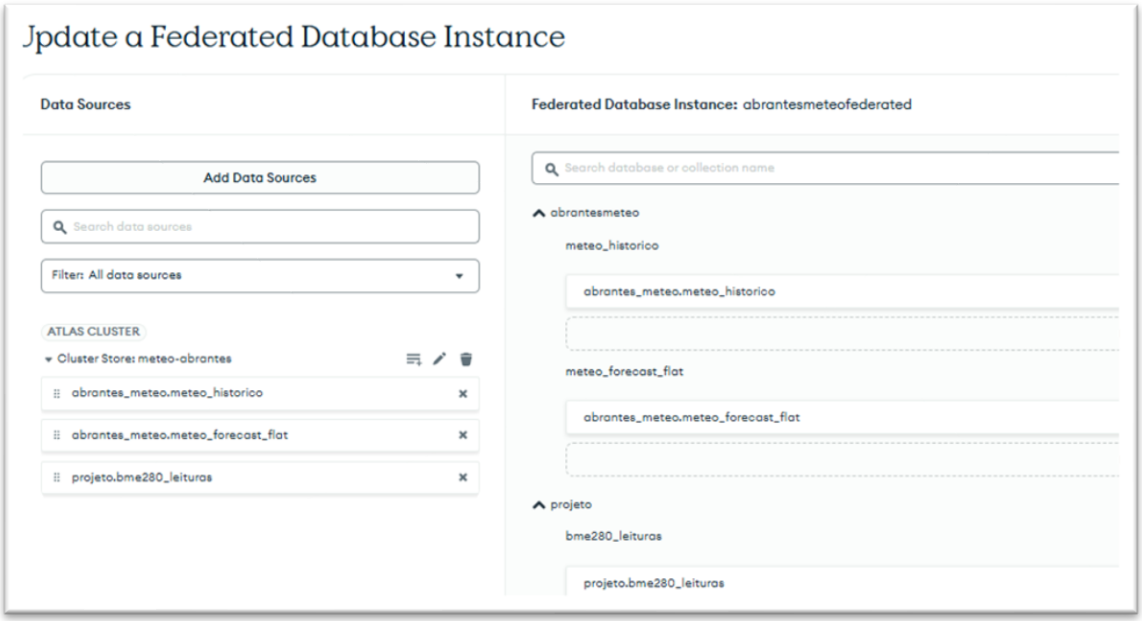


Figura 9 - Mapeamento das coleções do Atlas para a Federated Database Instance (Data Sources / Collections).

6.5.2. Geração do schema SQL (resolução do erro “no visible columns”)

Durante a ligação inicial ao Power BI foi identificado o erro “The table has no visible columns and cannot be queried”. A causa foi a ausência de schema SQL visível no Atlas SQL Interface para as coleções mapeadas. A correção consistiu em gerar os schemas por defeito na área de gestão de schemas da instância federada, passando as tabelas e colunas a ficar disponíveis para consulta.

Figura 10 - Ecrã ‘Manage SQL Schemas’ da instância federada - opção para gerar schemas por defeito.

Passos executados:

1. Atlas → Data Federation → selecionar a instância (abrantesmeteofederated).
2. Abrir “Manage SQL Schemas”.
3. Selecionar “Generate all default schemas”.
4. Aguardar a conclusão e confirmar que as tabelas/colunas passam a estar expostas no Atlas SQL.

6.5.3. Configuração do driver ODBC (DSN)

No Windows foi criado um **DSN ODBC** com o “MongoDB Atlas SQL ODBC Driver”, utilizando o endpoint fornecido pelo Atlas SQL para integração com Power BI. A ligação foi validada através do teste do DSN, confirmando conectividade e autenticação.

Figura 11 - Configuração do DSN ODBC para Atlas SQL/ODBC (nome do Data Source).

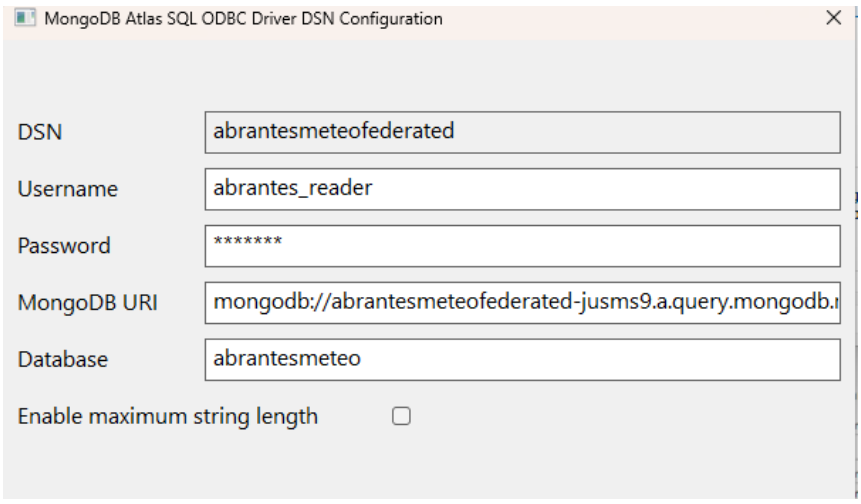


Figura 12 - Validação do DSN ODBC (teste de ligação bem-sucedido ao endpoint Atlas SQL).

6.6. Modelação no Power BI

Com a ligação operacional, o foco passou para a modelação: definição de tipo de dados, tabela calendário, relacionamentos e medidas analíticas. Foram mantidas as tabelas de origem (meteo_historico e meteo_forecast_flat) e criada uma tabela derivada (forecast_fixo) para suportar análise comparativa com baseline de previsão..

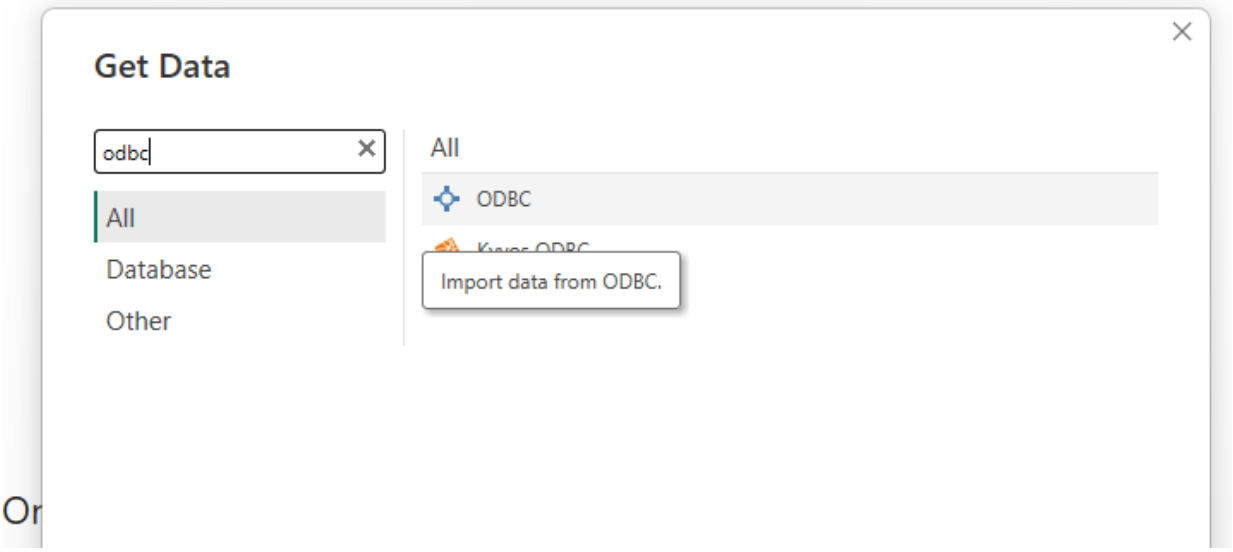


Figura 13 - Power BI Desktop: importação de dados através de ODBC (Atlas SQL).

6.6.1. Tabela calendário

Foi criada uma tabela calendário para garantir filtragem temporal consistente e suportar comparações entre séries temporais no mesmo eixo. A tabela contém uma coluna principal de data (sem hora) e atributos de apoio (ano, mês e dia) para segmentação e agregações.

```
Calendario =
ADDCOLUMNS (
    CALENDAR ( DATE(2023,1,1), DATE(2027,12,31) ),
    "Ano", YEAR ( [Date] ),
    "Mês", FORMAT ( [Date], "MMMM" ),
    "MêsNum", MONTH ( [Date] ),
    "Dia", DAY ( [Date] )
)
```

6.6.2. Relações (Model View)

A tabela calendário foi relacionada em **1:*** com as tabelas de factos através do campo de data. O desenho adotado evita relações diretas desnecessárias entre tabelas de Real e Previsto, garantindo que a comparação é conduzida por medidas e contexto temporal.

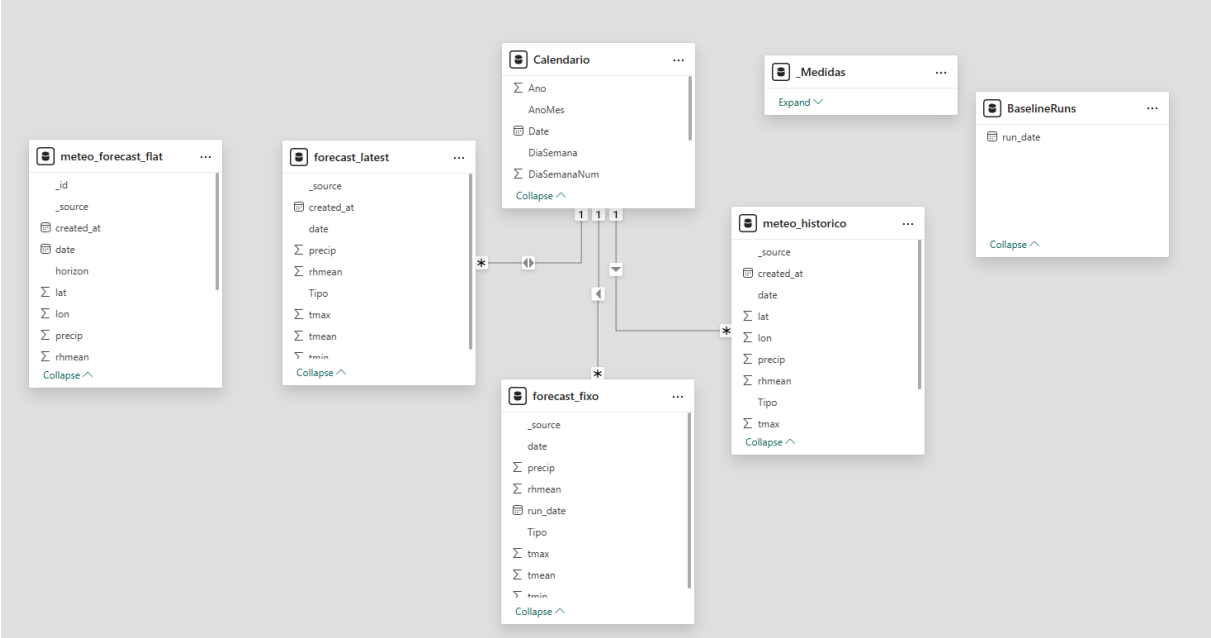


Figura 14 - Power BI: vista de modelo com tabelas de Real e Previsão para comparação.

6.6.3. forecast_fixo: baseline de previsão

Como a previsão é recalculada a cada execução, foi criada a tabela forecast_fixo para fixar uma execução escolhida como baseline. Esta abordagem permite comparar o que estava previsto com o que ocorreu, garantindo consistência analítica ao longo do tempo

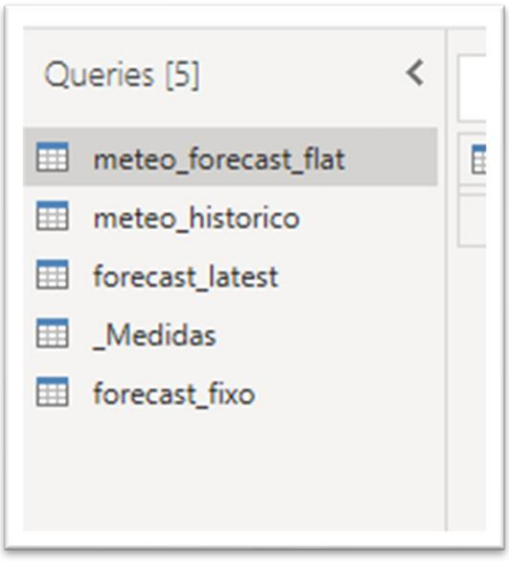


Figura 15 - Power Query: queries finais (tabelas base e tabelas derivadas para análise).

Tipos de dados recomendados (forecast_fixo):

Esta tabela é fundamental estar bem desenhada, pois é um clone da tabela forecast original e deve, para além de fixar os valores passados, receber os novos dados e assim poder fazer as análises comparativas. Neste projeto as colunas foram configuradas da seguinte forma:

date: Date (sem hora) - chave temporal.

run_date: Date - data da execução usada como baseline (constante na tabela).

created_at: Date/Time (opcional para auditoria) - útil se se pretender rastrear o momento exato da geração.

_source e Tipo: Text - identificadores de origem (prophet) e natureza (Previsto).

tmin, tmean, tmax, precip, rhmean, lat, lon: Decimal Number (ou Fixed decimal, se se quiser controlo de precisão).

_id, horizon, run_id: manter apenas se forem necessários; para análise diária básica, podem ser removidos no Power Query.

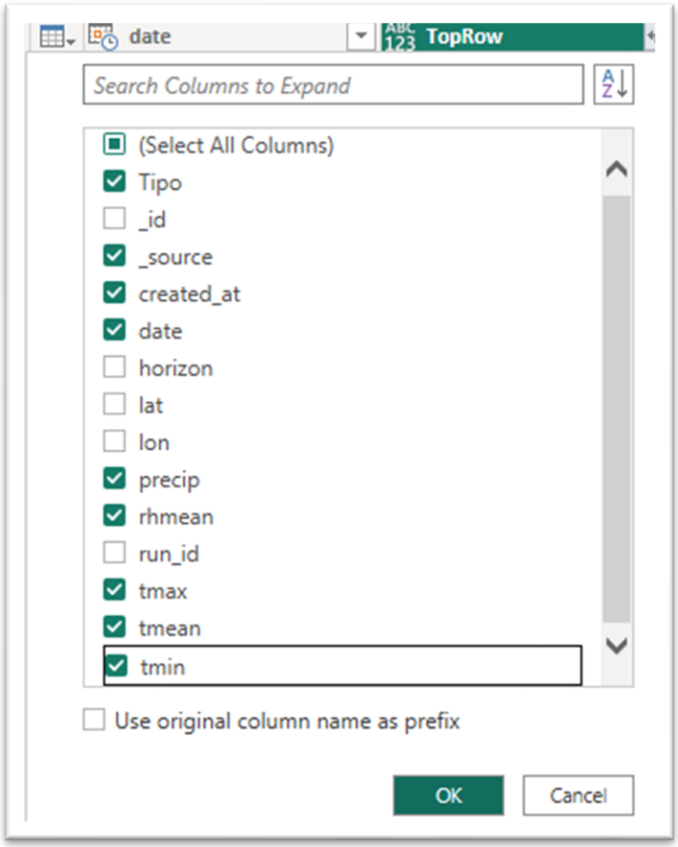


Figura 16 - Power Query: expansão de colunas ao normalizar registos (Expand record).

6.6.4. Medidas DAX para comparação (exemplo: Tmax, Tmin e Precip)

As medidas são criadas individualmente (uma a uma) no Power BI: Modelagem → Nova medida. A lógica base é: uma medida para o Real, outra para o Previsto (baseline), e uma medida de diferença.

```
-- Tmax
Real_Tmax =
AVERAGE ( meteo_historico[tmax] )

Prev_Tmax =
AVERAGE ( forecast_fixo[tmax] )

Dif_Tmax =
[Real_Tmax] - [Prev_Tmax]
```

```
-- Tmin
Real_Tmin =
AVERAGE ( meteo_historico[tmin] )

Prev_Tmin =
AVERAGE ( forecast_fixo[tmin] )

Dif_Tmin =
[Real_Tmin] - [Prev_Tmin]
```

```
-- Precipitação
Real_Precip =
AVERAGE ( meteo_historico[precip] )

Prev_Precip =
AVERAGE ( forecast_fixo[precip] )

Dif_Precip =
[Real_Precip] - [Prev_Precip]
```

Nota operacional:

Não é necessário aplicar filtros dentro das medidas para ‘run_date = baseline’, porque forecast_fixo já está filtrada no Power Query. Este desenho evita erros comuns no DAX (ex.: usar uma expressão True/False inválida como filtro de tabela). Isto levou a alfuns erros em fases iniciais pois não devolvia valores de forma lógica, ou pelo menos, na lógica pretendida.

6.7. Integração de Raspberry Pi 5 + sensor BME280

Este subcapítulo documenta a extensão do sistema Abrantes Meteo com um Raspberry Pi 5 equipado com um sensor BME280, permitindo recolher temperatura, humidade relativa e pressão atmosférica e disponibilizá-las no Power BI como uma tabela independente. A integração foi desenhada para ser simples, resiliente (execução autónoma via cron) e compatível com o modelo existente (MongoDB Atlas + ODBC + Power BI Service com Gateway).

6.7.1. Arquitetura e fluxo de dados

O fluxo implementado é o seguinte:

- Raspberry Pi lê o sensor BME280 via I2C;
- Raspberry executa um script Python que insere leituras numa coleção dedicada no MongoDB Atlas;
- Power BI Desktop consome os dados via ODBC (Atlas SQL Interface);
- dataset é publicado no Power BI Service e atualizado através do On-premises Data Gateway.

Foi mantida separação entre as leituras do BME280 e as coleções diárias (meteo_historico / meteo_forecast_flat) para evitar conflitos de granularidade (quase-real vs diário) e permitir visuais dedicados.

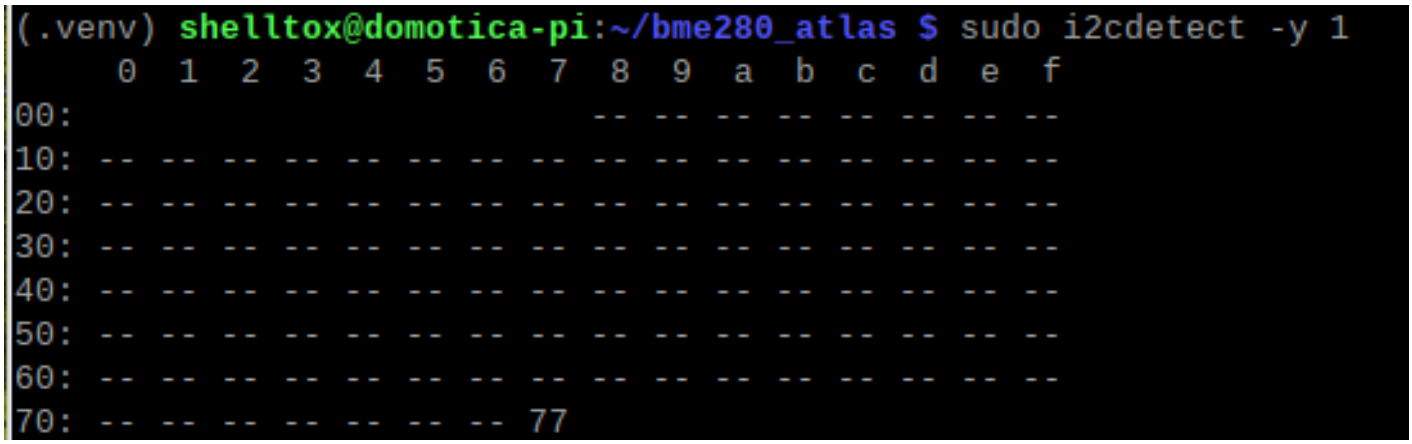


Figura 17 - Confirmação de endereço do sensor no Raspberry 0x77

6.7.2. Preparação do hardware e I2C no Raspberry Pi 5

O sensor BME280 foi ligado ao Raspberry Pi por I2C (alimentação 3.3V, GND, SDA e SCL) e o I2C foi ativado no sistema operativo. A deteção do sensor foi confirmada através do endereço I2C apresentado".

Comandos de validação (no Raspberry):

```
sudo apt update
sudo apt install -y i2c-tools
lsmod | grep i2c || true
sudo i2cdetect -y 1
```

6.7.3. Projeto Python no Raspberry (venv, dependências e ficheiro .env)

Foi criada uma pasta dedicada e um ambiente Python isolado para execução do script de leitura e envio. A ligação ao Atlas e parâmetros essenciais (base de dados, coleção e configuração do sensor) foram parametrizados num ficheiro .env local do projeto no Raspberry.
A validação local foi suportada por um script simples de leitura/diagnóstico.

Ver: **Anexo H — Raspberry Pi: read_local.py.**

Criação do ambiente e instalação de dependências (no Raspberry):

```
cd /home/shelltox
mkdir -p bme280_atlas && cd bme280_atlas
python3 -m venv .venv
source .venv/bin/activate
pip install --upgrade pip
# Bibliotecas Adafruit (I2C + BME280) e MongoDB
pip install pymongo dnspython adafruit-blinka adafruit-circuitpython-bme280
```

Configuração de variáveis de ambiente (.env). Este ficheiro concentra a ligação ao Atlas e os nomes da base de dados/coleção, evitando hardcode no código Python.
Configuração de variáveis de ambiente (.env). Este ficheiro concentra a ligação ao Atlas e os nomes da base de dados/coleção, evitando hardcode no código Python.

```
cat > .env <<'EOF'
# String de ligação copiada do Atlas (Data Federation / MongoDB Driver).
ATLAS_URI="mongodb://<db_user>:<db_pass>@<host>.a.query.mongodb.net/?ssl=true&authSource=a
dmin&appName=<appName>"

# Destino no Atlas
DB_NAME="projeto"
COL_NAME="bme280_leituras"

# Sensor
I2C_BUS="1"
I2C_ADDR="0x77"
EOF

# Carregar variáveis na sessão atual
set -a; source .env; set +a
```

Boas práticas aplicadas durante a resolução de erros de encoding: garantir locale UTF-8 e ficheiros em UTF-8. Em sistemas com locale “C” (ASCII), scripts e here-docs com acentos podem falhar com erros do tipo “Non-UTF-8 code …”. Uma mitigação simples é definir “LANG=C.UTF-8” e “LC_ALL=C.UTF-8” no ambiente de execução.

6.7.4. Script de recolha e envio para o Atlas

O envio para Atlas inclui:

- carimbo temporal UTC;
- identificação do equipamento e da origem;
- valores numéricos do sensor.

Este desenho permite integrar medições reais no ecossistema de análise e manter a coleção preparada para exposição via ODBC.

Ver: **Anexo J — Raspberry Pi: sensor_to_atlas.py.**

6.7.5. Automatização no Raspberry (cron: 5 leituras diárias)

Para garantir recolhas regulares sem intervenção manual, foi configurado um job de cron no utilizador “shelltox”. Neste cenário, são feitas 5 leituras por dia (07:00, 11:00, 15:00, 19:00 e 23:00).

Exemplo de crontab:

```
crontab -e

# 5 execuções diárias
0 7,11,15,19,23 * * * /home/shelltox/bme280_atlas/run_once.sh >>
/home/shelltox/bme280_atlas/cron.log 2>&1
```

Teste manual do script wrapper e criação do ficheiro de log:

```
/home/shelltox/bme280_atlas/run_once.sh
tail -n 50 /home/shelltox/bme280_atlas/cron.log
systemctl status cron --no-pager
```

Ver: **Anexo I — Raspberry Pi: run_once.sh.**

6.7.6. Monitorização local

Foi incluída monitorização básica para confirmar que o processo está operacional e para detetar rapidamente falhas (ausência de leituras recentes, falha de execução ou problemas de conectividade).
Ver: **Anexo G — Raspberry Pi: monitor_local.sh.**

6.7.7. Evidência de integração (Raspberry Pi → Atlas)

A integração é considerada válida quando:

- existem registos recentes no Atlas com a origem correta;
- timestamps e valores numéricos são plausíveis;
- o fluxo é repetível (novas execuções geram novos registos);
- a coleção é exposta no Atlas SQL/ODBC e carregável no Power BI, permitindo visuais temporais e indicadores de "última leitura".

Ver:

- **Anexo G — Raspberry Pi: monitor_local.sh**
- **Anexo H — Raspberry Pi: read_local.p**
- **Anexo I — Raspberry Pi: run_once.sh**
- **Anexo J — Raspberry Pi: sensor_to_atlas.py**

6.7.8. Atlas: permissões, coleção e schema para Power BI (Atlas SQL Interface)

Para o Power BI consumir esta coleção via ODBC (Atlas SQL Interface), a coleção tem de ter schema visível. Sem schema, o conector pode devolver o erro “The table has no visible columns and cannot be queried”.

Passos no MongoDB Atlas (via UI):

- Confirmar que a coleção existe e tem documentos: Cluster → Collections → DB “projeto” → collection “bme280_leituras”.
- Criar/usar um Database User com permissões mínimas (ex.: “read” no DB/collection, se for apenas leitura do Power BI; “readWrite” apenas para o Raspberry).
- Em Data Federation / SQL Interface: garantir que a base de dados e coleção estão mapeadas e expostas ao endpoint SQL/ODBC.
- Gerar/definir o schema da coleção para que os campos fiquem bem identificados e “visíveis” para consulta via ODBC.

Schema recomendado para “projeto.bme280_leituras” (tipos): “ts” (datetime), “source” (string), “host” (string), “temp_c” (double), “hum_rel” (double), “press_hpa” (double). Após aplicar o schema, a tabela passa a ser carregável no Power BI e desaparece os erros de colunas não visíveis.

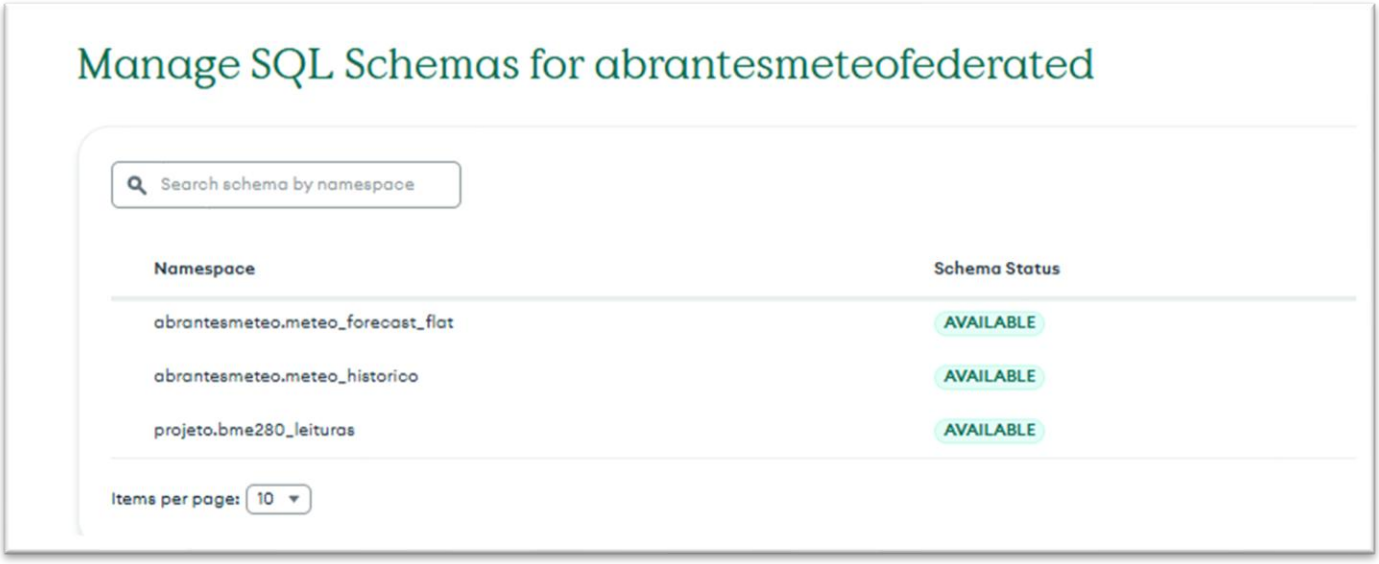


Figura 18 - SQL Schemas (Atlas Data Federation): namespaces disponíveis para consulta (ODBC/Power BI).

6.7.9. Power BI Desktop: importar a tabela e criar o scroller

1. **Importação**: Power BI Desktop → Get Data → ODBC → selecionar o DSN do Atlas (ex.: "abrantesmeteofederated"). No Navigator, escolher a base de dados "projeto" e a tabela "bme280_leituras" e carregar.
2. **Query "última leitura":** no Power Query Editor, duplicar "bme280_leituras" e chamar "bme280_latest" → ordenar por "ts" (descendente) → Keep Rows → Keep Top Rows → 1. Esta abordagem permite um cartão estável (sem DAX complexo) para mostrar a temperatura atual.
3. **Visual do valor atual:** inserir um Card (ou Card (New)) com o campo "temp_c" da query "bme280_latest". Formatar título (ex.: "Temperatura Atual - °C"), unidades e casas decimais.
4. **Scroller/zoom temporal:** criar um Line chart com "ts" no eixo X e "temp_c" em Values. Em Format → X-axis → ativar "Zoom slider" (funciona como scroller) para navegar em janelas temporais (ex.: últimas 24h, 7 dias, etc.).
5. **Publicação e atualização:** publicar o relatório no Power BI Service e configurar atualização programada usando o On-premises data gateway (personal mode) instalado no PC Desktop. O gateway é necessário porque o datasource é ODBC local, apesar de os dados estarem no Atlas.

Observação operacional: para um painel "quase em tempo real", o mais simples foi manter o cron no Raspberry (várias inserções/dia) e fazer refresh no Power BI Service com periodicidade compatível com o gateway e com as restrições do Power BI (ex.: de hora a hora). Se for necessário atualizar minuto-a-minuto, isso já entra em arquitetura de streaming (fora do âmbito do protótipo atual). Optei por manter o cron a correr no Raspberry 5 vezes ao dia e alinhar as atualizações do gateway para 1 hora após serem atualizadas via cron. E, assim, evitam-se alguns dissabores relacionados com diferenças horárias, já que o MongoDB Atlas, na versão gratuita, não disponibiliza o fuso horário de Portugal como opção.

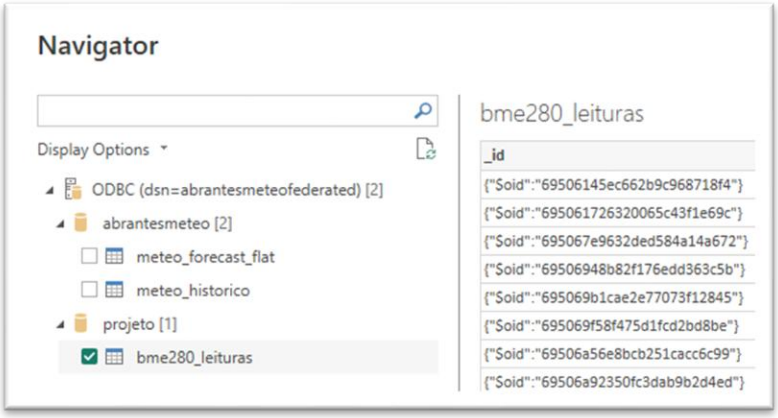


Figura 19 - Importação da Coleção "projeto.bme280_leituras" do MongoDB Atlas (pré-visualização de documentos e campos).

7. Dificuldades & Resolução

Um dos aspetos de mais relevo no projeto foi a resolução de problemas reais de integração. A seguir descrevem-se os incidentes mais relevantes, a causa provável e a ação corretiva aplicada.
Codificação UTF-8 no container Python: erros de execução quando o ficheiro não estava em UTF-8.

- Resolução: garantir gravação em UTF-8 e evitar caracteres inválidos.

Autenticação no Atlas (bad auth): erro por password/role incorretos.

- Resolução: recriar/confirmar utilizador no Database Access e validar com mongosh via mongodb+srv.

Power BI sem colunas visíveis: schema SQL não gerado no Atlas SQL.

- Resolução: Generate all default schemas em Manage SQL Schemas.

Colunas duplicadas (date e date.1) após expandir registos no Power Query:

- Resolução: remover date.1 e mudar rtipo de dados date como Date.

Erros de ligação ODBC/driver:

- Resolução por seleção do driver Atlas SQL ODBC e URI correta do Power BI Connector.

---

8. Resultados finais obtidos

No final, foi possível demonstrar:

- dados históricos carregados com sucesso;
- previsão diária gerada e persistida (365 dias por execução);
- sincronização para Atlas e exposição SQL/ODBC;
- dashboards no Power BI com comparação entre Real e Previsto, cada previsão tem um identificador de execução (run_id/run_date), e o previsto pode ser congelado para comparação à *posteriori.*

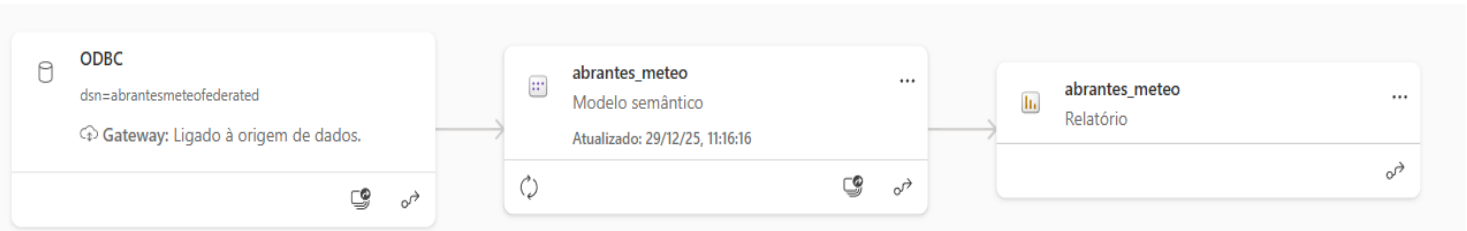


Figura 20 - Pipeline de publicação no Power BI Service: origem ODBC (Atlas SQL) com Gateway, dataset (modelo semântico) e relatório publicado.

9. Melhorias futuras e visão de evolução

Este projeto tem várias vias claras de evolução - e o mais relevante é que a arquitetura não é "meteorologia-dependente": pode ser reaproveitada para qualquer cenário de ingestão → armazenamento → transformação → previsão/analítica → visualização, como vendas, stocks, consumo energético, manutenção preventiva, tráfego, qualidade industrial, KPIs financeiros ou comportamento de utilizadores.

Principais melhorias e extensões (replicáveis noutras análises):

- Automatizar a criação de um forecast_fixo no servidor (coleção dedicada) para eliminar dependências do Power Query na fixação da base (garante consistência e repetibilidade do histórico de previsões).
- Persistir todas as execuções de forecast com run_id e gerar automaticamente métricas de desempenho (ex.: MAE/RMSE) por horizonte (D+1, D+7, D+30, etc.), criando uma base sólida para auditoria e melhoria contínua do modelo.
- Implementar observabilidade e alertas (Prometheus/Grafana e/ou logs estruturados com níveis e correlação por run_id) para detetar falhas, degradação de performance, atrasos de execução e anomalias operacionais.
- Reforçar gestão de segredos e segurança (Docker Secrets ou HashiCorp Vault) com rotação de credenciais do Atlas, princípio do menor privilégio e segregação de acessos por serviço.
- Adicionar camadas de qualidade de dados antes de persistir: validações, normalização, deteção de outliers, regras de consistência temporal e thresholds por variável (isto aplica-se tanto a sensores como a ERP/CRM/BI).
- Escalar para múltiplas fontes e múltiplas "entidades" (multi-cidade, multi-loja, multi-linha de produto, etc.) com parametrização e partição (por lat/lon, por entity_id, por período), mantendo custos e performance controlados.

Nota operacional crítica (não implementada, mas relevante):

Um ponto importante que ficou fora deste projeto foi garantir um ambiente Windows 24/7 em VM no servidor para suportar atualizações contínuas quando a camada de atualização depende de componentes Windows (ex.: gateways, refreshers, conectores específicos). A solução aplicada foi instalar e executar no PC e calendarizar atualizações em janelas horárias em que o equipamento supostamente está ligado. Para maturidade de produção, o ideal é mover essa dependência para infraestrutura sempre disponível (VM/serviço dedicado), reduzindo risco de falhas por indisponibilidade do posto local.

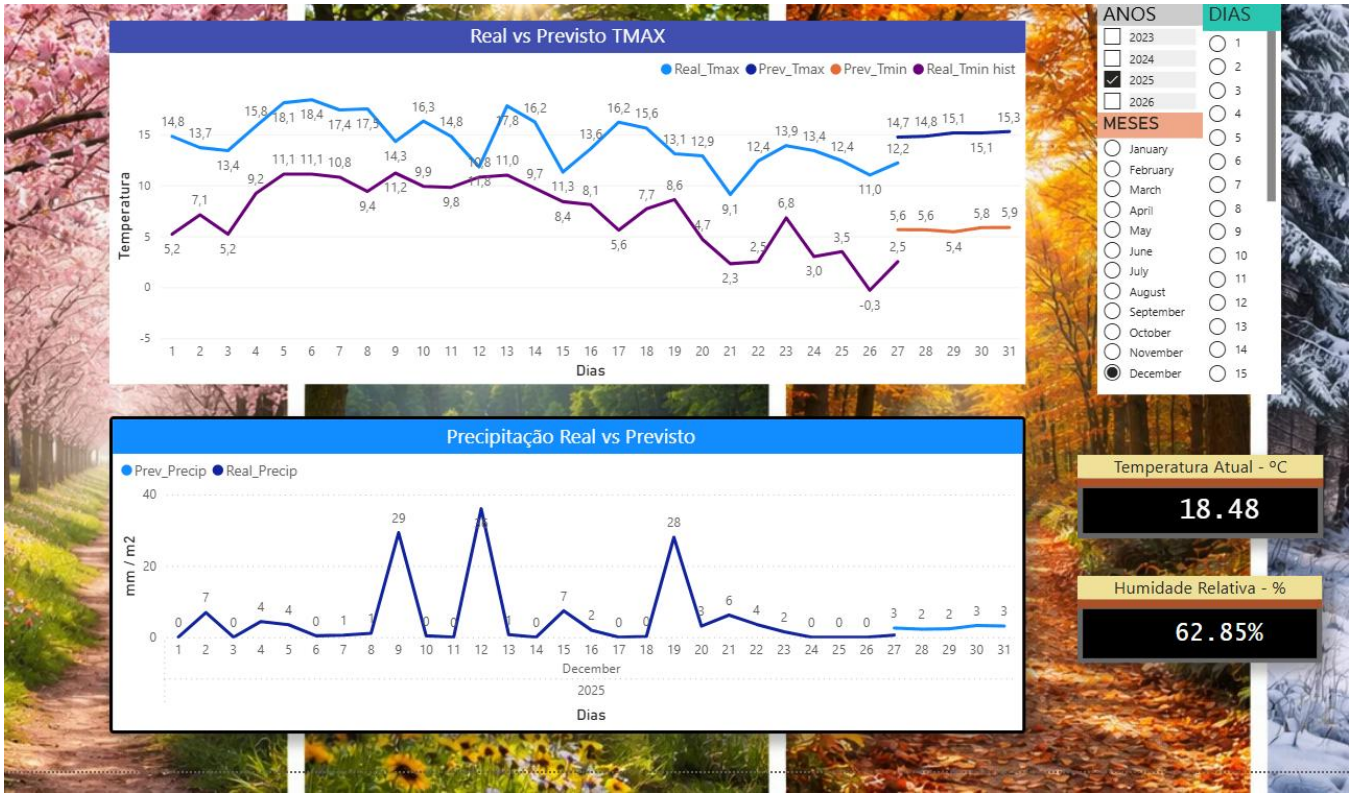


Figura 21 - Visual no Power BI: Cartões "Temperatura Atual, Humidade Relativa, Real vs Previsto em Temperatura e Precipitação".

10. Conclusão

O AbrantesMeteo resultou numa implementação operacional e coerente de um pipeline de dados completo, desde a recolha automática até à análise em Power BI, com armazenamento em MongoDB e previsão em Python com Prophet. A separação entre os dados reais e as previsões, bem como a definição de um baseline de forecast, permitiu que fosse possível efetuar comparações fiáveis e repetíveis ao longo do tempo, evitando leituras enviesadas por recalibrações diárias.

A opção por expor os dados via MongoDB Atlas e Atlas SQL (ODBC) foi determinante para garantir compatibilidade e estabilidade na sua utilização no Power BI, reduzindo dessa forma problemas típicos de integração com fontes NoSQL. A extensão com Raspberry Pi 5 e sensor BME280 adicionou uma componente experimental de relevo e também uma camada de complexidade, que apesar de à partida parecer de fácil integração se tenha revelado desafiante q.b., mas permitiu demonstrar a integração de medições físicas reais no mesmo ecossistema de dados.

No seu conjunto, o projeto permite validar uma abordagem modular e escalável, facilmente adaptável a outros contextos além da meteorologia (ex.: planeamento produção, vendas, stocks, energia ou manutenção), e evidencia competências práticas de integração de sistemas, automação e análise preditiva.

11. Glossário

API - Interface de programação que permite obter dados de um serviço externo (ex.: meteorologia) de forma automática.

Atlas (MongoDB Atlas) - Plataforma cloud gerida pela MongoDB para alojar bases de dados e disponibilizar serviços associados.

Atlas Data Federation - Camada do Atlas que permite virtualizar dados e expô-los como base virtual para consulta.

Atlas SQL (SQL Interface) - Interface SQL do Atlas (sobre Data Federation) para consultar dados MongoDB em formato tabular.

Backfill - Carregamento retroativo de histórico (ex.: desde 01-01-2023) para construir a base inicial de dados.

Baseline - Referência fixa usada para comparação (ex.: "previsão congelada" vs. "real observado").

BME280 - Sensor ambiental (temperatura, humidade e pressão atmosférica) usado no Raspberry Pi.

CMDSTANPY - Motor/biblioteca usada pelo Prophet para executar modelos (dependência do forecasting).

Coleção (MongoDB Collection) - Estrutura onde ficam guardados documentos (equivalente aproximado a "tabela" em SQL).

Container (Docker) - Unidade isolada que executa uma aplicação com dependências, de forma reprodutível.

Cron - Agendador no Linux para executar tarefas automaticamente em horários definidos.

DASHBOARD (Power BI) - Conjunto de visuais/indicadores para análise e acompanhamento.

Dataset / Modelo semântico (Power BI) - Camada de dados publicada no Power BI Service com modelo, relações e medidas.

DAX - Linguagem do Power BI para criar medidas/cálculos (ex.: Real vs. Previsto e diferenças).

Docker - Plataforma para construir e executar serviços em containers.

Docker Compose - Ficheiro/forma de orquestrar vários contentores e volumes (serviços do projeto).

Documento (MongoDB Document) - Registo em formato JSON/BSON (equivalente aproximado a "linha" em SQL).

DSN (ODBC Data Source Name) - Nome/configuração local do conector ODBC (host, credenciais, base, etc.).

Forecast - Previsão gerada pelo modelo para datas futuras (no projeto, horizonte de 365 dias).

Gateway (On-premises Data Gateway) - Componente do Power BI Service que permite refrescar fontes locais (ex.: ODBC/DSN).

I2C - Protocolo de comunicação usado para ligar o sensor BME280 ao Raspberry Pi.

Keyfile (MongoDB) - Ficheiro usado para autenticação interna do replica set quando a autenticação está ativa.

LXC - Tipo de contentor (a nível de sistema) usado no Proxmox para isolar o ambiente “docker-ct”.

Mongosh - Shell/cliente de linha de comandos para ligar ao MongoDB e validar queries/contagens.

MongoDB - Base de dados NoSQL orientada a documentos, usada para persistência local e em cloud.

ODBC - Standard de conectividade para aceder a dados via drivers, comum em ferramentas analíticas (Power BI).

Open-Meteo - Fonte/API de dados meteorológicos usada para histórico e atualização diária.

Power BI Desktop - Ferramenta local para modelação, Power Query, DAX e criação do relatório.

Power BI Service - Plataforma cloud onde o relatório/dataset é publicado e atualizado (refresh).

Power Query - Motor de transformação/limpeza de dados no Power BI (tipos, remoções, tabelas derivadas).

Proxmox - Hypervisor/plataforma do servidor (host) onde corre o contentor LXC.

Replica Set - Modo do MongoDB que suporta replicação e é frequentemente necessário para certas configurações com auth.

run_id / run_date - Identificador/data da execução diária do forecast, usado para rastreio e consistência analítica.

Schema (SQL Schema) - Definição de colunas/tipos expostos via Atlas SQL; sem isto podem surgir tabelas “sem colunas visíveis”.

Scheduler - Rotina que agenda a execução diária do pipeline (ingestão + forecast + sync).

Timezone / TZ - Configuração do fuso horário (ex.: Europe/Lisbon) para garantir execução e datas coerentes.

UTC - Referência de tempo padrão usada frequentemente em timestamps (evita ambiguidades de fusos horários).

WiredTiger - Motor de armazenamento do MongoDB (gera ficheiros internos como journal e metadata no volume).

12. Referências online

MongoDB. *Set Up and Query Data Federation – Atlas*. [Em linha]. Disponível em: https://www.mongodb.com/docs/atlas/data-federation/. [Consultado em: 19-10-2025]. Set Up and Query Data Federation - Atlas - MongoDB Docs

MongoDB. *Connect with ODBC Driver – Atlas (SQL Interface / Data Federation)*. [Em linha]. Disponível em: https://www.mongodb.com/docs/atlas/data-federation/query/sql/drivers/odbc/connect/. [Consultado em: 19-10-2025]. Connect with ODBC Driver - Atlas - MongoDB Docs

MongoDB. *Configure Database Users – Atlas (Database Access / Roles)*. [Em linha]. Disponível em: https://www.mongodb.com/docs/atlas/security-add-mongodb-users/. [Consultado em: 19-10-2025]. Configure Database Users - Atlas - MongoDB Docs

Microsoft. *MongoDB Atlas SQL interface connector (Power Query / Power BI)*. [Em linha]. Disponível em: https://learn.microsoft.com/en-us/power-query/connectors/mongodb-atlas-sql-interface. [Consultado em: 07-11-2025]. MongoDB Atlas SQL interface connector - Power Query | Microsoft Learn

Microsoft. *Use a Personal Gateway in Power BI (on-premises data gateway - personal mode)*. [Em linha]. Disponível em: https://learn.microsoft.com/en-us/power-bi/connect-data/service-gateway-personal-mode. [Consultado em: 07-11-2025]. Use a Personal Gateway in Power BI - Power BI | Microsoft Learn

Prophet (Meta). *Quick Start - Prophet documentation*. [Em linha]. Disponível em: https://facebook.github.io/prophet/docs/quick_start.html. [Consultado em: 14-12-2025]. Quick Start | Prophet

Adafruit. *Adafruit BME280 Library - Documentation (CircuitPython)*. [Em linha]. Disponível em: https://docs.circuitpython.org/projects/bme280/en/latest/. [Consultado em: 28-12-2025]. Introduction - Adafruit BME280 Library 1.0 documentation

13. Anexos

No servidor (docker)

Anexo A - docker-compose.yml

Orquestra serviços (MongoDB, bootstrap e aplicação) com dependências e reinício automático.

BLOCO 1 - Serviço mongodb

Função: Sobe o MongoDB local com autenticação e ReplicaSet, persistindo dados e expondo uma porta no host.

```
mongodb:
  image: mongo:7
  container_name: abrantes_meteo_mongodb
  command: ["--bind_ip_all","--replSet","rs0","--keyFile","/etc/mongo-keyfile","--auth"]
  environment:
    MONGO_INITDB_ROOT_USERNAME: ${MONGO_ROOT_USERNAME}
    MONGO_INITDB_ROOT_PASSWORD: ${MONGO_ROOT_PASSWORD}
  ports:
    - "27018:27017"   # 27018 no host para não colidir com outros projetos
  volumes:
    - ./mongo-data:/data/db
    - ./secrets/mongo-keyfile:/etc/mongo-keyfile:ro

  healthcheck:
    test: ["CMD-SHELL", "mongosh --quiet -u $$MONGO_INITDB_ROOT_USERNAME -p
$$MONGO_INITDB_ROOT_PASSWORD --authenticationDatabase admin --eval
'db.adminCommand({ping:1})' || exit 1"]
    interval: 10s
    timeout: 5s
    retries: 30
    start_period: 20s
  restart: unless-stopped
```

Notas:
ReplicaSet (rs0) + keyFile: necessário para consistência e configuração do utilizador da app.
Volume ./mongo-data: garante persistência do histórico e forecasts.
healthcheck: permite dependências robustas (a app só arranca após o Mongo estar operacional).

BLOCO 2 - Serviço mongo_setup

Função: Executa uma única vez o script de bootstrap do MongoDB (ReplicaSet, user app e índices).

```
mongo_setup:
  image: mongo:7
  container_name: abrantes_meteo_mongo_setup
  depends_on:
    mongodb:
      condition: service_healthy
```

```
  env_file: .env
  volumes:
    - ./mongo-init:/mongo-init:ro
  command: ["bash","-lc","mongosh
mongodb://$MONGO_ROOT_USERNAME:$MONGO_ROOT_PASSWORD@mongodb:27017/admin --file /mongo-
init/01-setup.js"]
  restart: "no"
```

Notas:
Corre apenas após o Mongo estar healthy (evita falhas no setup).
restart: "no" para não repetir; o setup deve ser idempotente mas não precisa correr sempre.

BLOCO 3 - Serviço app

Função: Constrói e executa a aplicação Python que faz backfill, forecast e sync diário.

```
app:
  build:
    context: ./app
  container_name: abrantes_meteo_app
  env_file: .env
  depends_on:
    mongodb:
      condition: service_healthy
    mongo_setup:
      condition: service_completed_successfully
  restart: unless-stopped
```

Notas:
Depende de mongodb + mongo_setup: garante que a BD está pronta e com índices antes da app correr.
restart: unless-stopped: recupera automaticamente após falhas de energia/reboot.

Anexo B - .env

BLOCO 1 - Fuso horário e localização

Função: Define o fuso horário e as coordenadas usadas nas chamadas à API, bem como a data inicial do histórico.

```
TZ=Europe/Lisbon
LAT=39.46
LON=-8.20
START_DATE=2023-01-01
```

Notas:
TZ afeta o scheduler e a interpretação da API (timezone).
LAT/LON definem o ponto geográfico (Abrantes).
START_DATE define o início do backfill (histórico).

BLOCO 2 - Hora de execução diária

Função: Configura a hora local em que o ciclo diário deve correr.

```
RUN_HOUR_LOCAL=7
```

Notas:
Define quando a app corre: backfill (se necessário) + forecast + sync.

BLOCO 3 - Credenciais e nomes do projeto

Função: Define utilizadores/passwords e nomes lógicos da base de dados e conta da aplicação.

```
MONGO_ROOT_USERNAME=root
MONGO_ROOT_PASSWORD=<SECRET>

APP_DB=abrantes_meteo
APP_USER=app
APP_PASSWORD=<SECRET>
```

Notas:
MONGO_ROOT_* é usado no bootstrap (01-setup.js).
APP_USER/APP_PASSWORD é usado pela aplicação para readWrite na DB do projeto.
No anexo, passwords foram mascaradas por segurança.

BLOCO 4 - URIs de ligação (Local e Atlas)

Função: Define endpoints de ligação do Mongo local (dentro do Docker) e do MongoDB Atlas (cloud).

```
LOCAL_URI="mongodb://app:<SECRET>@mongodb:27017/abrantes_meteo?authSource=abrantes_meteo&repli
caSet=rs0&authMechanism=SCRAM-SHA-256"
ATLAS_URI="mongodb+srv://abrantes_writer:<SECRET>@meteo-
abrantes.vt8gavx.mongodb.net/abrantes_meteo?retryWrites=true&w=majority&appName=meteo-
abrantes"
```

Notas:
LOCAL_URI usa o hostname do serviço Docker: mongodb.

ATLAS_URI é usada apenas na fase de sync (Power BI consome via Atlas SQL/ODBC).

BLOCO 5 - Parâmetros operacionais

Função: Controla robustez da sincronização e horizonte do forecast.

```
SYNC_LOOKBACK_DAYS=3
FORECAST_HORIZON_DAYS=365
```

Notas:
SYNC_LOOKBACK_DAYS reenvia N dias para mitigar falhas/retries.
FORECAST_HORIZON_DAYS controla quantos dias o Prophet prevê.

Anexo C - requirements.txt

Lista de dependências Python com versões fixas (reprodutibilidade).

BLOCO 1 - HTTP (API)

Função: Dependência para chamadas à API Open-Meteo.

```
requests==2.32.5
```

Notas:
Usada na ingestão para obter dados diários.

Função: Driver oficial para ler/escrever em MongoDB com bulk operations.

```
pymongo==4.15.5
```

Notas:
Suporta UpdateOne + bulk_write e upsert (idempotência).

BLOCO 3 - Processamento de dados

Função: Estruturas DataFrame e manipulação de datas para treino/previsão.

```
pandas==2.2.3
```

Notas:
Organiza histórico em série temporal (ds/y) para o Prophet.

BLOCO 4 - Modelo preditivo

Função: Biblioteca Prophet para forecast diário.

```
prophet==1.1.6
```

Notas:
Gera yhat para horizonte configurado, separando Real vs Previsto por run_id.

BLOCO 5 - Backend do Prophet

Função: Runtime necessário para o Prophet executar (Stan via cmdstanpy).

```
cmdstanpy==1.2.5
```

Notas:
Melhora estabilidade do Prophet em ambientes de contentor.

Anexo D - mongo-init/01-setup.js

Bootstrap do MongoDB: ReplicaSet, utilizador da app e índices (idempotente).

BLOCO 1 - Inicialização do ReplicaSet

Função: Garante que o MongoDB está em ReplicaSet (rs0); se não estiver, inicializa.

```
try {
  const st = rs.status();
  print("ReplicaSet ja ativo:", st.set);
} catch (e) {
  rs.initiate({
    _id: "rs0",
    members: [{ _id: 0, host: "mongodb:27017" }]
  });
}
```

Notas:
ReplicaSet é usado para coerência da configuração e compatibilidade com keyFile/auth.

BLOCO 2 - waitPrimary()

Função: Espera até o nó ficar PRIMARY antes de criar user/índices (evita falhas por estado transitório).

```
function waitPrimary() {
  for (let i = 0; i < 60; i++) {
    try {
      const st = rs.status();
      const primary = st.members && st.members.find(m => m.stateStr === "PRIMARY");
      if (primary) return true;
    } catch(e) {}
    sleep(1000);
  }
  return false;
}

if (!waitPrimary()) {
  throw new Error("ReplicaSet nao ficou PRIMARY a tempo.");
}
```

Notas:
Sem PRIMARY, operações administrativas podem falhar intermitentemente.

BLOCO 3 - Ler variáveis do ambiente

Função: Obtém APP_DB, APP_USER e APP_PASSWORD vindos do .env.

```
const dbName = process.env.APP_DB || "abrantes_meteo";
const user   = process.env.APP_USER || "app";
const pwd    = process.env.APP_PASSWORD;
```

Notas:
Evita hardcode e permite reconfiguração por ambiente.

BLOCO 4 - Criar utilizador da aplicação (idempotente)

Função: Cria o utilizador de aplicação apenas se não existir.

```
const adb = db.getSiblingDB(dbName);

const existing = adb.getUser(user);
if (!existing) {
  adb.createUser({
    user: user,
    pwd: pwd,
    roles: [{ role: "readWrite", db: dbName }]
  });
} else {
  print("User ja existia:", user);
}
```

Notas:
readWrite apenas na DB do projeto: princípio do menor privilégio.

BLOCO 5 - Criar índices (idempotente)

Função: Cria índices essenciais para integridade e performance do histórico e forecasts.

```
adb.meteo_historico.createIndex({ date: 1 }, { unique: true });

adb.meteo_forecast_flat.createIndex({ date: 1 });
adb.meteo_forecast_flat.createIndex({ run_id: 1, date: 1 }, { unique: true });
adb.meteo_forecast_flat.createIndex({ created_at: -1 });
adb.meteo_forecast_flat.createIndex({ Tipo: 1 });
adb.meteo_forecast_flat.createIndex({ horizon: 1 });
```

Notas:
meteo_historico(date unique) -> um registo por dia (Real).
meteo_forecast_flat(run_id+date unique) -> não duplica previsões por execução.
created_at/Tipo/horizon -> acelera filtros no Power BI/Atlas SQL.

Anexo E - app/Dockerfile

Construção da imagem da app (artefacto imutável) com dependências controladas.

BLOCO 1 - Base image

Função: Seleciona uma imagem Python minimalista para reduzir tamanho e superfície de ataque.

```
FROM python:3.11-slim
```

Notas:
Versão fixa (3.11) facilita reprodutibilidade.

BLOCO 2 - Dependências do sistema

Função: Instala pacotes mínimos necessários e limpa cache para reduzir a imagem final.

```
RUN apt-get update && apt-get install -y --no-install-recommends \
    tzdata ca-certificates curl procps \
 && rm -rf /var/lib/apt/lists/*
```

Notas:
tzdata garante timezone correto no container.
rm -rf /var/lib/apt/lists/* reduz tamanho da imagem.

BLOCO 3 - Diretório de trabalho

Função: Define /app como diretório onde o código é executado.

```
WORKDIR /app
```

Notas:
Padroniza caminhos internos para COPY e CMD.

BLOCO 4 - Instalar dependências Python

Função: Copia o requirements e instala versões fixas.

```
COPY requirements.txt /app/requirements.txt
RUN pip install --no-cache-dir -r /app/requirements.txt
```

Notas:
--no-cache-dir reduz tamanho; requirements fixos aumentam reprodutibilidade.

BLOCO 5 - Copiar aplicação e comando de arranque

Função: Coloca o app.py na imagem e define o comando default.

```
COPY app.py /app/app.py
CMD ["python","/app/app.py"]
```

Notas:
O container arranca sempre a mesma aplicação (artefacto imutável).

Anexo F - app.py (pipeline completo)

App.py: pipeline (ingestão → persistência → previsão → sync → agendamento)

Implementa o ciclo diário do AbrantesMeteo:
(1) obtém dados diários da API Open-Meteo;
(2) guarda/atualiza no MongoDB local por upsert;
(3) treina Prophet e grava forecast com run_id;
(4) sincroniza para Atlas;
(5) agenda execução diária.

Estruturas de dados na BD:
• Coleção meteo_historico (Real): date UTC (00:00), Tipo='Real', tmean/tmax/tmin/precip/rhmean, created_at, _source, lat/lon.
• Coleção meteo_forecast_flat (Previsto): date UTC (00:00), Tipo='Previsto', horizon='365d', run_id (YYYY-MM-DD), created_at, _source, lat/lon, variáveis previstas.

BLOCO 1 - Imports e dependências

Função: Carregar bibliotecas (HTTP, datas, pandas, MongoDB e Prophet).

```
import os
import time
import json
import math
import requests
import pandas as pd
from datetime import datetime, timedelta, date, timezone
from zoneinfo import ZoneInfo
from pymongo import MongoClient, ASCENDING, UpdateOne
from prophet import Prophet
```

Notas:
requests → chamada à API Open-Meteo.
pandas → preparação de séries temporais para Prophet.
pymongo → upserts e bulk writes (eficiência + idempotência).
zoneinfo → gestão correta de fusos horários.

BLOCO 2 - Configuração por variáveis de ambiente (parametrização)

Função: Tornar o comportamento configurável sem alterar código.

```
TZ = os.getenv("TZ", "Europe/Lisbon")
LOCAL_TZ = ZoneInfo(TZ)

LAT = float(os.getenv("LAT", "39.46"))
LON = float(os.getenv("LON", "-8.20"))
START_DATE = os.getenv("START_DATE", "2023-01-01")

RUN_HOUR_LOCAL = int(os.getenv("RUN_HOUR_LOCAL", "7"))
FORECAST_HORIZON_DAYS = int(os.getenv("FORECAST_HORIZON_DAYS", "365"))

LOCAL_URI = os.getenv("LOCAL_URI")
ATLAS_URI = os.getenv("ATLAS_URI", "").strip()
APP_DB = os.getenv("APP_DB", "abrantes_meteo")
SYNC_LOOKBACK_DAYS = int(os.getenv("SYNC_LOOKBACK_DAYS", "3"))
```

Notas:
RUN_HOUR_LOCAL: hora diária do ciclo (scheduler).
FORECAST_HORIZON_DAYS: horizonte do forecast (365 dias).
SYNC_LOOKBACK_DAYS: janela de segurança para re-enviar histórico (mitiga falhas).

BLOCO 3 - Mapeamento de variáveis (API → campos internos)

Função: Normalizar nomes da API para nomes "limpos" no Mongo/Power BI.

```
DAILY_VARS = {
    "temperature_2m_mean": "tmean",
    "temperature_2m_max": "tmax",
    "temperature_2m_min": "tmin",
    "precipitation_sum": "precip",
    "relative_humidity_2m_mean": "rhmean",
}
```

Notas:
Evita nomes longos no Mongo/Power BI e padroniza o modelo de dados.

BLOCO 4 - Utilitários de logging e datas

Função: Centralizar logging e garantir datas consistentes (UTC midnight).

```
def log(msg: str):
    print(msg, flush=True)

def today_local() -> date:
    return datetime.now(LOCAL_TZ).date()

def yesterday_local() -> date:
    return today_local() - timedelta(days=1)

def to_utc_midnight(d: date) -> datetime:
    return datetime(d.year, d.month, d.day, tzinfo=timezone.utc)
```

Notas:
to_utc_midnight evita problemas de comparação/join por datas e mantém a BD consistente.

BLOCO 5 - Ligações MongoDB (local e Atlas) + índices

Função: Criar clientes MongoDB e impor índices essenciais (performance + integridade).

```
def mongo_local() -> MongoClient:
    return MongoClient(LOCAL_URI)

def mongo_atlas() -> MongoClient | None:
    if not ATLAS_URI:
        return None
    return MongoClient(ATLAS_URI)

def ensure_indexes(db):
    db.meteo_historico.create_index([("date", ASCENDING)], unique=True)
    db.meteo_forecast_flat.create_index([("date", ASCENDING)])
    db.meteo_forecast_flat.create_index([("run_id", ASCENDING), ("date", ASCENDING)],
unique=True)
    db.meteo_forecast_flat.create_index([("created_at", -1)])
    db.meteo_forecast_flat.create_index([("Tipo", ASCENDING)])
```

```
    db.meteo_forecast_flat.create_index([("horizon", ASCENDING)])
```

Notas:
meteo_historico(date unique) → um registo por dia (Real).
meteo_forecast_flat(run_id+date unique) → não duplica previsões de uma execução.

BLOCO 6 - Ingestão da API Open-Meteo (diário)

Função: Chamar a API (intervalo de datas) e transformar resposta em documentos.

```
def fetch_open_meteo_daily(start: date, end: date) -> list[dict]:
    if end < start:
        return []
    url = "https://archive-api.open-meteo.com/v1/archive"
    params = {
        "latitude": LAT,
        "longitude": LON,
        "start_date": start.isoformat(),
        "end_date": end.isoformat(),
        "daily": ",".join(DAILY_VARS.keys()),
        "timezone": TZ
    }
    r = requests.get(url, params=params, timeout=60)
    r.raise_for_status()
    data = r.json()
    daily = data.get("daily", {})
    times = daily.get("time", [])
    out = []
    for i, t in enumerate(times):
        d = date.fromisoformat(t)
        doc = {
            "date": to_utc_midnight(d),
            "Tipo": "Real",
            "horizon": None,
            "run_id": None,
            "created_at": datetime.now(timezone.utc),
            "_source": "open-meteo-archive",
            "lat": LAT,
            "lon": LON,
        }
        for api_key, field in DAILY_VARS.items():
            arr = daily.get(api_key, [])
            doc[field] = float(arr[i]) if i < len(arr) and arr[i] is not None else None
        out.append(doc)
    return out
```

Notas:
Constrói o pedido com latitude/longitude e variáveis.
Para cada dia, cria um documento “Real” com campos normalizados.

BLOCO 7 - Escrita idempotente (upsert) e backfill em blocos

Função: Garantir histórico completo e atualizado sem duplicações.

```
def upsert_historico(db, docs: list[dict]) -> int:
    if not docs:
        return 0
    ops = []
    for doc in docs:
```

```
        ops.append(UpdateOne({"date": doc["date"]}, {"$set": doc}, upsert=True))
    res = db.meteo_historico.bulk_write(ops, ordered=False)
    return (res.upserted_count or 0) + (res.modified_count or 0) + (res.matched_count or 0)

def backfill_historico(db):
    last = db.meteo_historico.find({}, {"date": 1}).sort("date", -1).limit(1)
    last_doc = next(last, None)
    start = date.fromisoformat(START_DATE)
    if last_doc and last_doc.get("date"):
        last_date = last_doc["date"].astimezone(timezone.utc).date()
        start = max(start, last_date + timedelta(days=1))

    end = yesterday_local()
    if end < start:
        log(f"[historico] jaatualizado ate{end.isoformat()}")
        return

    log(f"[historico] backfill de {start.isoformat()} ate {end.isoformat()}")

    block = 60
    d0 = start
    while d0 <= end:
        d1 = min(end, d0 + timedelta(days=block-1))
        docs = fetch_open_meteo_daily(d0, d1)
        upsert_historico(db, docs)
        log(f"[historico] upsert {d0.isoformat()}..{d1.isoformat()} ({len(docs)} dias)")
        d0 = d1 + timedelta(days=1)
```

Notas:
Calcula o start com base no último registo existente.
Usa blocos de 60 dias para estabilidade e robustez da API.
bulk_write + upsert=True → rápido e repetível (idempotente).

BLOCO 8 - Previsão com Prophet (horizonte 365 dias)

Função: Treinar um modelo por variável e gravar previsões com run_id.

```
def build_forecast_365(db) -> tuple[str, int]:
    cur = list(db.meteo_historico.find({}, {"_id": 0, "date": 1, "tmean": 1, "tmax": 1,
"tmin": 1, "precip": 1, "rhmean": 1}))
    if not cur:
        log("[forecast] sem historico para treinar.")
        return ("", 0)

    df = pd.DataFrame(cur)
    df["date"] = pd.to_datetime(df["date"], utc=True).dt.date
    df = df.sort_values("date")

    last_real = df["date"].max()
    start_fc = last_real + timedelta(days=1)
    end_fc = last_real + timedelta(days=FORECAST_HORIZON_DAYS)
    dates = pd.date_range(start_fc, end_fc, freq="D")

    run_id = today_local().isoformat()
    created_at = datetime.now(timezone.utc)

    preds = {}

    def fit_var(var: str):
        s = df[["date", var]].dropna()
        if s.empty:
            return None
        series = s.rename(columns={"date": "ds", var: "y"})
```

```
        series["ds"] = pd.to_datetime(series["ds"])
        m = Prophet(daily_seasonality=True, weekly_seasonality=True,
yearly_seasonality=True)
        m.fit(series)
        fut = pd.DataFrame({"ds": pd.to_datetime(dates)})
        fc = m.predict(fut)[["ds", "yhat"]]
        return fc

    for var in ["tmean", "tmax", "tmin", "precip", "rhmean"]:
        fc = fit_var(var)
        if fc is None:
            log(f"[forecast] WARN sem dados para {var}")
            continue
        fc["ds"] = fc["ds"].dt.date
        preds[var] = fc.set_index("ds")["yhat"]

    ops = []
    n = 0
    for d in dates.date:
        doc = {
            "date": to_utc_midnight(d),
            "Tipo": "Previsto",
            "horizon": "365d",
            "run_id": run_id,
            "created_at": created_at,
            "_source": "prophet",
            "lat": LAT,
            "lon": LON,
            "tmean": float(preds["tmean"].get(d)) if "tmean" in preds and
pd.notna(preds["tmean"].get(d)) else None,
            "tmax":  float(preds["tmax"].get(d))  if "tmax"  in preds and
pd.notna(preds["tmax"].get(d))  else None,
            "tmin":  float(preds["tmin"].get(d))  if "tmin"  in preds and
pd.notna(preds["tmin"].get(d))  else None,
            "precip":float(preds["precip"].get(d))if "precip"in preds and
pd.notna(preds["precip"].get(d))else None,
            "rhmean":float(preds["rhmean"].get(d))if "rhmean"in preds and
pd.notna(preds["rhmean"].get(d))else None,
        }
        ops.append(UpdateOne({"run_id": run_id, "date": doc["date"]}, {"$set": doc},
upsert=True))
        n += 1

    if ops:
        db.meteo_forecast_flat.bulk_write(ops, ordered=False)

    log(f"[forecast] run_id={run_id} docs={n}
({start_fc.isoformat()}..{end_fc.isoformat()})")
    return (run_id, n)
```

Notas:
Lê histórico, define intervalo futuro, treina Prophet por variável.
Grava yhat por dia em meteo_forecast_flat.
run_id identifica a execução do dia (auditoria e comparações).

BLOCO 9 - Sincronização para MongoDB Atlas

Função: Enviar para a cloud o que interessa ao Power BI: real recente + forecast do dia.

```
def sync_to_atlas(local_db):
    atlas_cli = mongo_atlas()
    if atlas_cli is None:
        log("[sync] ATLAS_URI vazio, sync ignorado (ok para testes locais).")
```

```
        return

    atlas_db = atlas_cli[APP_DB]
    ensure_indexes(atlas_db)

    end = yesterday_local()
    start = end - timedelta(days=SYNC_LOOKBACK_DAYS)
    start_dt = to_utc_midnight(start)
    end_dt = to_utc_midnight(end) + timedelta(days=1)

    hist_docs = list(local_db.meteo_historico.find(
        {"date": {"$gte": start_dt, "$lt": end_dt}},
        {"_id": 0}
    ))

    if hist_docs:
        ops = [UpdateOne({"date": d["date"]}, {"$set": d}, upsert=True) for d in hist_docs]
        atlas_db.meteo_historico.bulk_write(ops, ordered=False)
        log(f"[sync] historico enviado para Atlas: {len(hist_docs)} docs
({start.isoformat()}..{end.isoformat()})")
    else:
        log("[sync] historico: nada para enviar (janela lookback vazia).")

    rid = today_local().isoformat()
    fc_docs = list(local_db.meteo_forecast_flat.find({"run_id": rid}, {"_id": 0}))
    if fc_docs:
        ops = [UpdateOne({"run_id": d["run_id"], "date": d["date"]}, {"$set": d},
upsert=True) for d in fc_docs]
        atlas_db.meteo_forecast_flat.bulk_write(ops, ordered=False)
        log(f"[sync] forecast enviado para Atlas: {len(fc_docs)} docs (run_id={rid})")
    else:
        log(f"[sync] forecast: sem docs para run_id={rid} (pode ser normal se ainda nao
ocorreu hoje).")

    atlas_cli.close()
```

Notas:
Reenvia últimos N dias → “rede de segurança” contra falhas.
Envia apenas o run_id do dia → reduz volume e simplifica consumo.

BLOCO 10 - Orquestração de um ciclo completo

Função: Executar a sequência correta (índices → backfill → forecast → sync).

```
def run_once():
    cli = mongo_local()
    db = cli[APP_DB]
    ensure_indexes(db)

    backfill_historico(db)
    build_forecast_365(db)
    sync_to_atlas(db)

    cli.close()
```

Notas:
Centraliza o fluxo principal: prepara BD, atualiza histórico, prevê e sincroniza.

BLOCO 11 - Scheduler (cálculo do próximo run + loop principal)

Função: Correr uma vez ao arrancar e depois diariamente à hora definida.

```
def next_run_time_local() -> datetime:
    now = datetime.now(LOCAL_TZ)
    target = now.replace(hour=RUN_HOUR_LOCAL, minute=0, second=0, microsecond=0)
    if target <= now:
        target = target + timedelta(days=1)
    return target

def main():
    log(f"[app] TZ={TZ} lat={LAT} lon={LON} start={START_DATE} run_hour={RUN_HOUR_LOCAL}")
    log("[app] primeira execucao(backfill + forecast + sync)")
    run_once()

    while True:
        nxt = next_run_time_local()
        log(f"[scheduler] next_run={nxt.isoformat()} tz={TZ}")
        while datetime.now(LOCAL_TZ) < nxt:
            time.sleep(30)
        log("[scheduler] executar ciclo diario")
        try:
            run_once()
        except Exception as e:
            log(f"[ERROR] ciclo diario falhou: {e}")
            time.sleep(10)

if __name__ == "__main__":
    main()
```

Notas:
Faz “primeira execução” ao arrancar (útil após reboot/falha de energia).
Ciclo diário resiliente: apanha exceções e não termina o processo.

Anexo G - Raspberry Pi: monitor_local.sh

Monitorização local contínua do BME280 (debug), sem envio para a cloud.

BLOCO 1 - Contexto do projeto e ambiente virtual

Função: Entra na pasta do projeto e ativa a virtualenv onde estão as dependências do sensor.

```
cd /home/shelltox/bme280_atlas || exit 1
source .venv/bin/activate
```

Notas:
Evita conflitos com Python global do Raspberry.

BLOCO 2 - Carregar variáveis de ambiente

Função: Carrega .env para que scripts tenham parâmetros (ex.: I2C_ADDR).

```
set -a; source .env; set +a
```

Notas:
set -a exporta automaticamente variáveis para subprocessos.

BLOCO 3 - Loop de leitura local

Função: Imprime leituras a cada 5 segundos, útil para validar o sensor sem enviar para Atlas.

```
while true; do
  python read_local.py
  sleep 5
done
```

Notas:
Modo 'monitor' para debug local e calibração.

Anexo H - Raspberry Pi: read_local.py

Leitura local do BME280 e impressão de valores (usado pelo monitor_local.sh).

BLOCO 1 - Imports e driver do sensor

Função: Carrega bibliotecas do barramento I2C e driver Adafruit BME280.

```
import board, busio
import adafruit_bme280.basic as adafruit_bme280
```

Notas:
Necessário para comunicar via I2C e ler temperatura/humidade/pressão.

BLOCO 2 - read_once()

Função: Inicializa I2C, cria o objeto BME280 e devolve as três medições arredondadas.

```
def read_once(addr=0x77):
    i2c = busio.I2C(board.SCL, board.SDA)
    bme = adafruit_bme280.Adafruit_BME280_I2C(i2c, address=addr)
    temp_c = round(float(bme.temperature), 2)
    hum_rel = round(float(bme.relative_humidity), 2)
    press_hpa = round(float(bme.pressure), 2)
    return temp_c, hum_rel, press_hpa
```

Notas:
Arredondamento a 2 casas melhora apresentação e reduz ruído de medição.

BLOCO 3 - main()

Função: Lê I2C_ADDR do ambiente e imprime uma linha com host e medições.

```
addr = int(os.getenv("I2C_ADDR","0x77"), 16)
host = socket.gethostname()
t,h,p = read_once(addr)
print(f"host={host} addr={hex(addr)} temp_c={t} hum_rel={h} press_hpa={p}")
```

Notas:
Permite confirmar rapidamente leituras e endereço I2C.

Anexo I - Raspberry Pi: run_once.sh

Wrapper de execução única para cron: prepara ambiente e chama sensor_to_atlas.py.

BLOCO 1 - Fail fast

Função: Força erro imediato se algum comando falhar.

```
set -e
```

Notas:
Evita execuções incompletas sem erro (importante em cron).

BLOCO 2 - Contexto + venv

Função: Entra no diretório e ativa o ambiente virtual.

```
cd /home/shelltox/bme280_atlas
source .venv/bin/activate
```

Notas:
Garante que usa as dependências corretas (Adafruit/PyMongo).

BLOCO 3 - Carregar .env

Função: Exporta variáveis necessárias para ligar ao Atlas e definir DB/coleção.

```
set -a
source .env
set +a
```

Notas:
Mantém configuração fora do código (boas práticas).

BLOCO 4 - Execução do envio

Função: Corre o script que lê o sensor e insere no MongoDB Atlas.

```
python sensor_to_atlas.py
```

Notas:
Este é o comando que deve ser chamado pelo cron.

Anexo J - Raspberry Pi: sensor_to_atlas.py

Leitura do BME280 e inserção no MongoDB Atlas (timestamp UTC, índice em ts).

BLOCO 1 – Imports

Função: Carrega bibliotecas de ambiente/tempo, MongoDB e drivers do sensor.

```
import os
import socket
from datetime import datetime, timezone
from pymongo import MongoClient, DESCENDING
import board
import busio
import adafruit_bme280.basic as adafruit_bme280
```

Notas:
`DESCENDING` é usado para índice por timestamp (última leitura rápida).

BLOCO 2 - must_env()

Função: Valida que variáveis críticas existem (ex.: ATLAS_URI).

```
def must_env(name: str) -> str:
    v = os.getenv(name)
    if not v:
        raise SystemExit(f"Missing env var: {name}")
    return v
```

Notas:
Falha explícita é preferível a erros silenciosos em cron.

BLOCO 3 - read_bme280()

Função: Lê temperatura, humidade e pressão via I2C.

```
def read_bme280(i2c_addr: int):
    i2c = busio.I2C(board.SCL, board.SDA)
    bme = adafruit_bme280.Adafruit_BME280_I2C(i2c, address=i2c_addr)
    return float(bme.temperature), float(bme.humidity), float(bme.pressure)
```

Notas:
I2C_ADDR configurável permite usar 0x76/0x77 conforme o módulo.

BLOCO 4 - main(): configuração

Função: Lê parâmetros (DB, coleção, source, endereço I2C) e valida ATLAS_URI.

```
atlas_uri = must_env("ATLAS_URI")
dbn = os.getenv("DB_NAME", "projeto")
coln = os.getenv("COL_NAME", "bme280_leituras")
source = os.getenv("SOURCE", "raspi-bme280")
i2c_addr = int(os.getenv("I2C_ADDR", "0x77"), 16)
```

Notas:
DB_NAME/COL_NAME permitem separar ambientes sem alterar código.

BLOCO 5 - main(): documento e escrita no Atlas

Função: Cria documento com `ts` UTC e insere na coleção no Atlas, garantindo índice por timestamp.

```
temp_c, hum_rel, press_hpa = read_bme280(i2c_addr)

ts = datetime.now(timezone.utc)
doc = {
    "ts": ts,
    "source": source,
    "host": socket.gethostname(),
    "temp_c": round(temp_c, 2),
    "hum_rel": round(hum_rel, 2),
    "press_hpa": round(press_hpa, 2),
}

cli = MongoClient(atlas_uri, serverSelectionTimeoutMS=8000)
c = cli[dbn][coln]
c.create_index([("ts", DESCENDING)])
c.insert_one(doc)
cli.close()
```

Notas:
ts em UTC facilita joins e comparações no Power BI.
Índice em ts (desc) acelera 'última leitura' (Top 1).

Anexo L - Organograma Inicial após 1ª revisão

CTeSP INF - UC PROJETO INTEGRADO 2025/26			Semana actual:	1			
PROJETO ForecastLab ABT			INICIO	FINAL	% CUMPRIMENTO	DIAS RESTANTES	NOTAS / OBSERVAÇÕES
			#####	16-dez-25	31%	-15	O projeto ForecastLabABT visa criar uma plataforma integrada para a cidade de Abrantes que combina dados meteorológicos históricos (10 anos), recolha em tempo real através de websites públicos e de um sensor local, e a construção de modelos preditivos de curto prazo (7-10 dias), sazonais (1-3 meses) e de longo prazo (10 anos), permitindo comparar previsões com observações reais, identificar desvios e disponibilizar resultados, acessíveis local e remotamente, apoiando decisões de planeamento com base em informação meteorológica fiável e visualmente clara.
OBJETIVOS	FERRAMENTAS	RESULTADO ESPERADO					
00 — Definição do Projeto - Propôr projeto a desenvolver durante o	**Documento de requisitos, Proxmox, D**	**Âmbito, objetivos e arquitetura defi**	#####	30-set-25	100%	OK	
Definir âmbito, objetivos e entregáveis	Word/Excel	Documento inicial			100%	OK	
Preparar arquitetura técnica (UM880, Pi, Power BI)	Proxmox, Raspberry Pi, Power BI	Infraestrutura planeada			100%	OK	
Criar repositório inicial e diretórios no servidor	Docker	Estrutura base criada			100%	OK	
01 - API & Monitorização	**Flask, Docker, Cloudflare**	**API pública segura**	1-out-25	14-out-25	33%	-78	
Configurar segurança básica (token + Cloudflare Tunnel) - criar dominio	Cloudflare, HTTPS	Acesso remoto seguro			100%	OK	.json onde inclua historico 2 anos, acrescentar dados diarios, juntar 5 localidades de Portugal, dados estatisticos, tendências e previsões a futuro
Criar API Flask (criar serviço web com páginas que devolvem dados necessários (histórico, tempo real, etc)	Flask, Python	API funcional			100%	OK	
Criar página de teste para garantir que a API e base de dados estão a funcionar	Flask	Monitorização simples			0%		
Validar tempos de resposta e fiabilidade	cURL, logs	Testes OK			0%		
02 — Carregar dados Tempo Real (3×/dia)	**Python, Docker, Raspberry Pi**	**Dados contínuos em tempo real**	#####	29-out-25	50%	-63	
Criar script para recolha de dados públicos	Python, cron	Inserção 3×/dia em Mongo			100%	OK	
Configurar Raspberry Pi 5 com sensor BME280 para recolha de dados em tempo rea localmente	Raspberry Pi, Python BME280	Leituras locais 3×/dia			0%		
Criar jobs automáticos em Docker com scheduler	Docker, cron	Pipelines automatizados			0%		
Validar inserção de dados em MongoDB, consultas simples para confirmação de dados	MongoDB	Dados confirmados			100%	OK	
03 — Dados & ETL Histórico	**Python, Pandas, MongoDB**	**Histórico 10 anos carregado**	#####	12-nov-25	33%	-49	
Recolher datasets meteorológicos (IPMA, NOAA, ECMWF)	APIs públicas, Python	Ficheiros CSV/JSON			100%		
Normalizar formatos e converter para MongoDB	Python ETL, MongoDB	Dados uniformizados			0%		
Criar índices por timestamp para otimizar consultas	MongoDB	Consultas rápidas			0%		
04 — Modelos Preditivos	**Python (Prophet, ARIMA), Pandas**	**Modelos operacionais**	#####	26-nov-25	0%	-35	
Modelo curto prazo (7-10 dias)	Prophet/ARIMA	Forecast diário			0%		
Previsões sazonais (1-3 meses, percentis/anomalias)	Python, estatística	Seasonal outlook			0%		
Climatologia/tendências (10 anos)	Python, MongoDB	Séries climatológicas			0%		
Validar resultados com métricas	Python, Scikit-learn	Relatório de precisão			0%		
05 — Dashboards Power BI	**Power BI, MongoDB**	**Dashboards interativos**	#####	4-dez-25	0%	-27	
Criar dashboard de histórico (10 anos)	Power BI	Série temporal completa			0%		
Criar dashboard de tempo real (web + sensor)	Power BI	Leituras atualizadas			0%		
Criar dashboard de previsões (curto prazo + sazonal + clima)	Power BI	Visualização preditiva			0%		
Testar exportações (Excel/PDF)	Power BI	Funcionalidade validada			0%		
06 — Qualidade & Relatório Final	**Word, PowerPoint, Power BI**	**Projeto entregue**	5-dez-25	16-dez-25	0%	-15	
Rever métricas de precisão e ingestão	Python, MongoDB	Dados e previsões validadas			0%		dia 13 entrega e dia 16 defesa
Redigir relatório académico com anexos técnicos	Word	Relatório em PDF			0%		
Preparar apresentação e demo remota (UM880 acessível)	PowerPoint, Cloudflare	Ensaio final			0%		
Ensaiar defesa e validação final	Apresentação ao docente	Projeto aprovado			0%		

Anexo M - Organograma projeto final

(revisto e ajustado à realidade para comparação com a previsão inicial)

CTeSP INF - UC PROJETO INTEGRADO 2025/26				Semana actual:	1			
PROJETO abrantes_meteo	**AÇÕES**			**INICIO**	**FINAL**	**% CUMPRIMENTO**	**DIAS RESTANTES**	**NOTAS / OBSERVAÇÕES**
				16-set-25	6-jan-26	95%	8	Inicialmente projetei um sistema que serviria para prever as necessidades de produção e custos de uma organização. Com base na análise de dados históricos e previsionais, permitiria construir modelos preditivos para estimar necessidades de produção, recolheria e compararia essa informação com dados em tempo real, por forma a contruir uma estrutura de informação de apoio aos gestores na tomada de decisões com dashboards e alertas.
OBJETIVOS	**TAREFAS**	**FERRAMENTAS**	**RESULTADO ESPERADO**					
O0 — Enquadramento e Arquitetura do Projeto				**16-set-25**	**19-set-25**	**100%**	**OK**	
Definir requisitos, objetivo e arquitetura (Docker -> Atlas -> Power BI -> Web)	Âmbito e objetivos do sistema; modelo de dados (Real/Previsto); periodicidade (recolha de dados); cadeia de consumo (Atlas SQL/ODBC -> Power BI) e critérios de validação. Ferramentas: MS Word/diagramas, MongoDB Atlas, Power BI. Resultado esperado: arquitetura validada e plano de execução.			16-set-25	19-set-25	100%	OK	Definição do projeto final e ferramentas a utilizar para obter o resultado pretendido
O1 — Infraestrutura Local (Docker-ct) e MongoDB				**22-set-25**	**10-out-25**	**100%**	**OK**	
Criar estrutura do projeto e ficheiros base (.env, pastas, keyfile)	Criar pastas e subpastas na doker - Resultado esperado: infraestrutura local reprodutível, com volumes persistentes e Mongo saudável.	Debian Linux; Docker Compose; OpenSSL; MongoDB 7	Infraestrutura local operacional (MongoDB com replicaset/auth) e pronta para ingestão.	22-set-25	26-set-25	100%	OK	Estrutura criada no docker-ct, na pasta /mnt/storage/abrantes_meteo, num servidor local com Proxmox a correr uma versão de Linux Debian, onde correm os scripts necessários para a ingestaõ de dados inicias bem como as atualizações diárias planificadas
Configurar docker-compose (MongoDB 7 + setup + app) e volumes persistentes				29-set-25	3-out-25	100%	OK	
Deploy local, healthcheck e resolução de falhas (keyfile/permissões/replicaset)				6-out-25	10-out-25	100%	OK	
O2 — Ingestão, Normalização e Previsão (Python + Prophet)				**13-out-25**	**7-nov-25**	**100%**	**OK**	
Implementar ingestão diária (Open-Meteo) e upsert em meteo_historico	Desenvolver app/app.py: importar histórico (desde 01-01-2023), uniformizar campos, gerar previsões (365 dias) e guardar por execução (run_id).	Python 3.11; requests; pandas; pymongo; Prophet/cmdstanpy	Coleções localmente preenchidas (histórico + forecast) e prontas para sincronizar.	13-out-25	17-out-25	100%	OK	
Implementar previsão (Prophet) 365 dias e persistência em meteo_forecast_flat				20-out-25	31-out-25	100%	OK	
Agendar execução diária, logs e correções (encoding/dependências)				3-nov-25	7-nov-25	100%	OK	
O3 — Persistência Cloud (Atlas) e Sincronização				**10-nov-25**	**14-nov-25**	**100%**	**OK**	
Criar cluster no MongoDB Atlas e configurar acesso (Network + Users)	Configurar MongoDB Atlas: criar cluster e utilizadores, permitir IP, ligar via ATLAS_URI (mongosh) e sincronizar histórico + previsões.	MongoDB Atlas; mongosh; Docker logs	Persistência cloud operacional (meteo_historico e meteo_forecast_flat) com sync diário.	10-nov-25	10-nov-25	100%	OK	Criação de cluster e coleções. Importação dos dados do mongo local para Atlas, criação de utilizadores. Paralelamente instalação do Power Bi Connector e driver Mongo Atlas ODBC, por forma a configurar o DSN e assim lifar ao Power BI
Configurar ATLAS_URI no .env e sincronização (lookback + run_id)				13-nov-25	13-nov-25	100%	OK	
Validar integridade (contagens e últimos registos) local vs Atlas				13-nov-25	14-nov-25	100%	OK	
O4 — Exposição SQL/ODBC (Data Federation + Atlas SQL)				**17-nov-25**	**21-nov-25**	**100%**	**OK**	
Criar Data Federation e mapear coleções do cluster	Criar Data Federation, mapear coleções e gerar SQL Schemas; configurar ODBC/DSN (VirtualDatabase) e validar ligação no Power BI	Atlas Data Federation; Atlas SQL; ODBC Driver	Tabelas expostas via SQL/ODBC com colunas e tipos reconhecidos pelo Power BI.	17-nov-25	21-nov-25	100%	OK	
Gerar SQL Schemas (Atlas SQL) para expor colunas ao ODBC/Power BI				17-nov-25	21-nov-25	100%	OK	
Configurar driver ODBC + DSN e validar consulta no Power BI Navigator				17-nov-25	21-nov-25	100%	OK	
O5 — Power BI (Modelo, DAX, Dashboards e Publicação Web)				**24-nov-25**	**26-dez-25**	**100%**	**OK**	
Importar dados no Power BI via ODBC e normalizar tipos no Power Query	Power BI: importar via ODBC, tratar no Power Query, criar medidas DAX (Real vs Previsto), publicar e configurar refresh + embed público.	Power BI; Power Query; DAX; ODBC; Gateway; HTML/CSS/JS	Dashboards (Real vs Previsto) publicados e prontos para consulta/refresh.	24-nov-25	26-dez-25	100%	OK	Importar coleção do Atlas e normalizar os tabelas no power query, criação de medidas paar usar nos dashboards
Modelar (calendário e relações) e criar medidas DAX (Real vs Previsto)				24-nov-25	26-dez-25	100%	OK	
Dashboards, publicação, refresh e integração web (embed Power BI)				24-nov-25	26-dez-25	100%	OK	
O6 — Validação Final, Documentação e Defesa				**27-dez-25**	**3-jan-26**	**67%**	5	
Testes end-to-end e verificação do scheduler/refresh	Validação final; consolidar documentação/anexos e preparar apresentação/demonstração.	Docker; mongosh; MS Word/PDF;Browser	Entrega final: pipeline operacional + documentação completa + defesa preparada.	27-dez-25	29-dez-25	100%	OK	A apresentação será feita via browser numa página criada para o efeito e onde estão incluidos os dashboards do power bi publicados via web e sem restrições de login
Relatório académico e anexos (ficheiros, prints e evidências)				29-dez-25	30-dez-25	100%	OK	
Apresentação e defesa (website + demonstração)				3-jan-26	6-jan-26	0%	NÃO INICIADA	

Politécnico de Tomar
Escola Superior de Tecnologia de Abrantes

www.ipt.pt