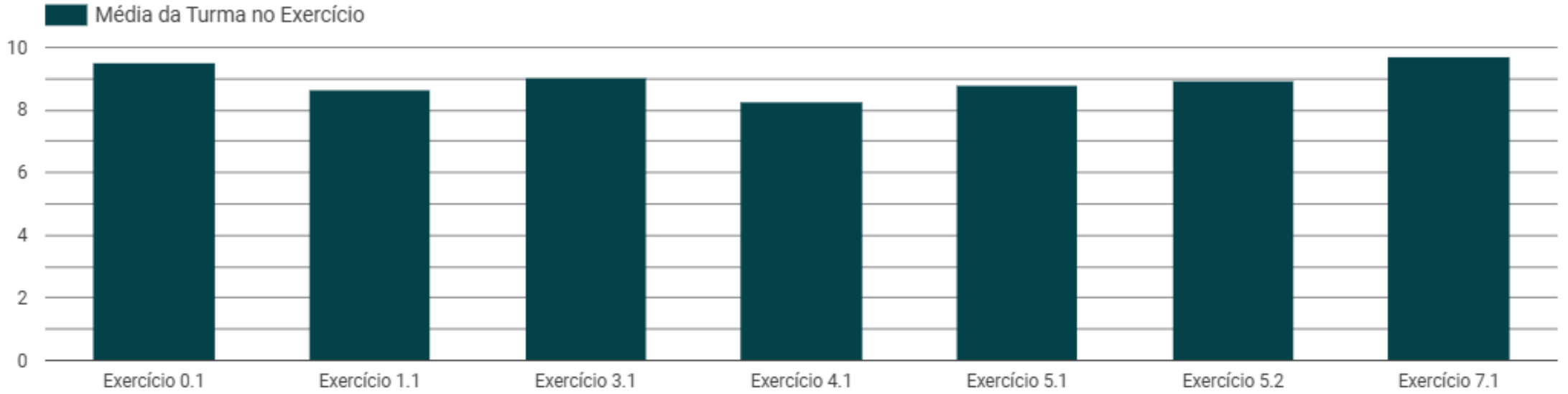
Arquitetura e Engenharia de Dados

Professor: Alex Pereira

Desempenho da Turma



Motivação: Agendamento de Tarefas no Cron (linux)

- Configuração do cron (commando **`crontab -e`**)

```
# m h  DoM mon DoW   command
* * * * * echo "Hello World $(date)" >> $HOME/cron_demo.txt
* * * * * /usr/bin/python3 /home/vboxuser/coleta_dolar.py
```

```
vboxuser@ubuntu2:~$ cat ./cron_demo.txt
Hello World Sun Aug 31 01:01:01 PM UTC 2025
Hello World Sun Aug 31 01:02:01 PM UTC 2025
Hello World Sun Aug 31 01:03:01 PM UTC 2025
Hello World Sun Aug 31 01:04:01 PM UTC 2025
Hello World Sun Aug 31 01:05:01 PM UTC 2025
```

```
vboxuser@ubuntu2:~$ cat ./cotacao_dolar.txt
2025-08-31 13:35:01 - Cotação do dólar: R$ 5.4274
2025-08-31 13:36:02 - Cotação do dólar: R$ 5.4274
2025-08-31 13:37:01 - Cotação do dólar: R$ 5.4274
2025-08-31 13:38:01 - Cotação do dólar: R$ 5.4274
2025-08-31 13:39:02 - Cotação do dólar: R$ 5.4274
```

Problemas com Agendamento com Cron

- Confiabilidade / Re-tentativas (retries)
- Rastreabilidade e logs
- Encadeamento de múltiplas tarefas
- Backfill e recuperação
- Paralelismo e escalabilidade
- Monitoramento e alertas
- Governança e padrões
  - Pipelines em Código python, versionamento, testes

Apache
Airflow

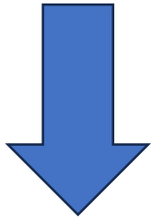
Orquestrador de workflows e pipelines

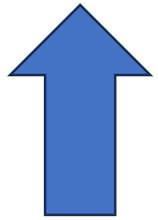
Útil para manter um fluxo automatizado e recorrente de dados

Dashboard do Preço do Bitcoin

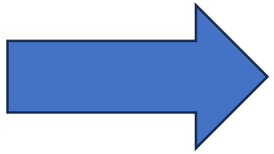
coingecko

Looker Studio





Apache
Airflow



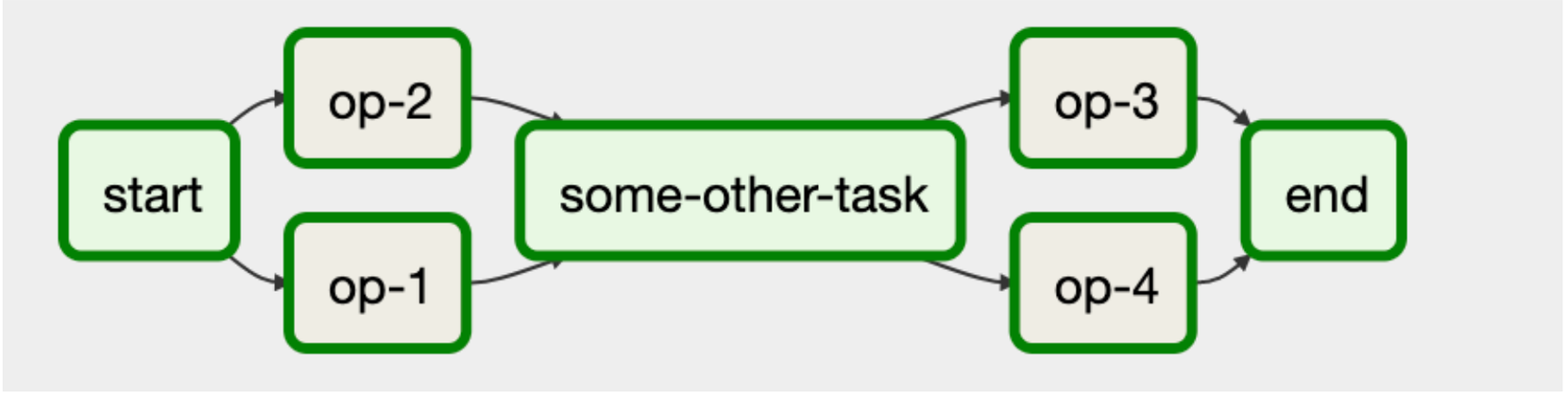
PostgreSQL

BigQuery

Dashboard do Preço do Bitcoin

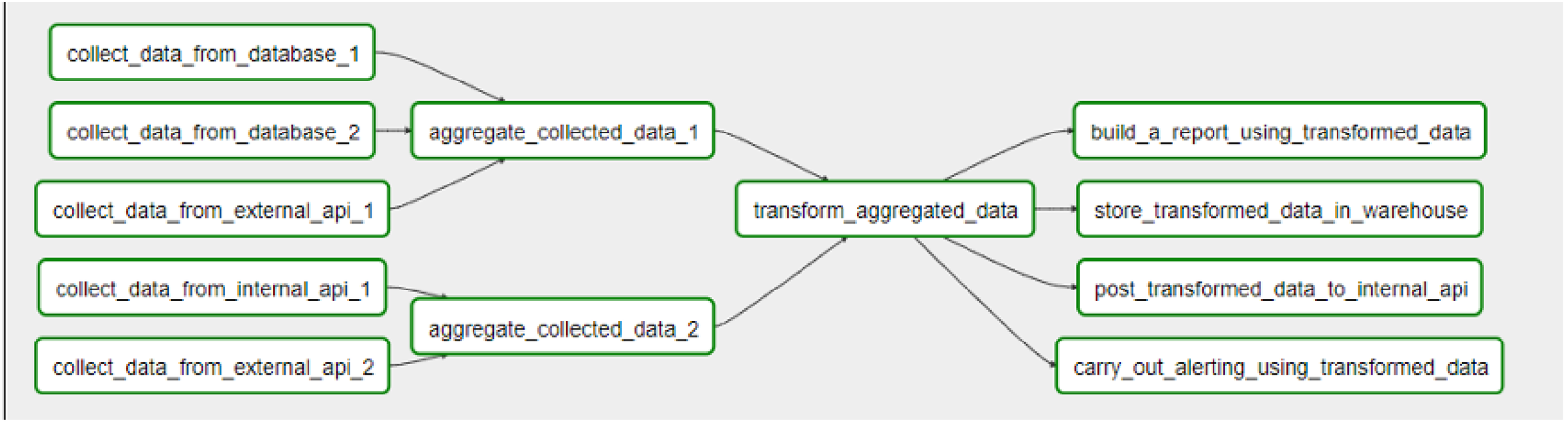
DAG (Directed Acyclic Graph)

- Grafo acíclico direto
  - Sem laços
    - ✓ Tarefas sequenciais e paralelizáveis
- Operators
  - Building blocks do Airflow
    - ✓ Contém a lógica / implementação dos requisitos; e
    - ✓ Templates prontos pra configurar e usar.



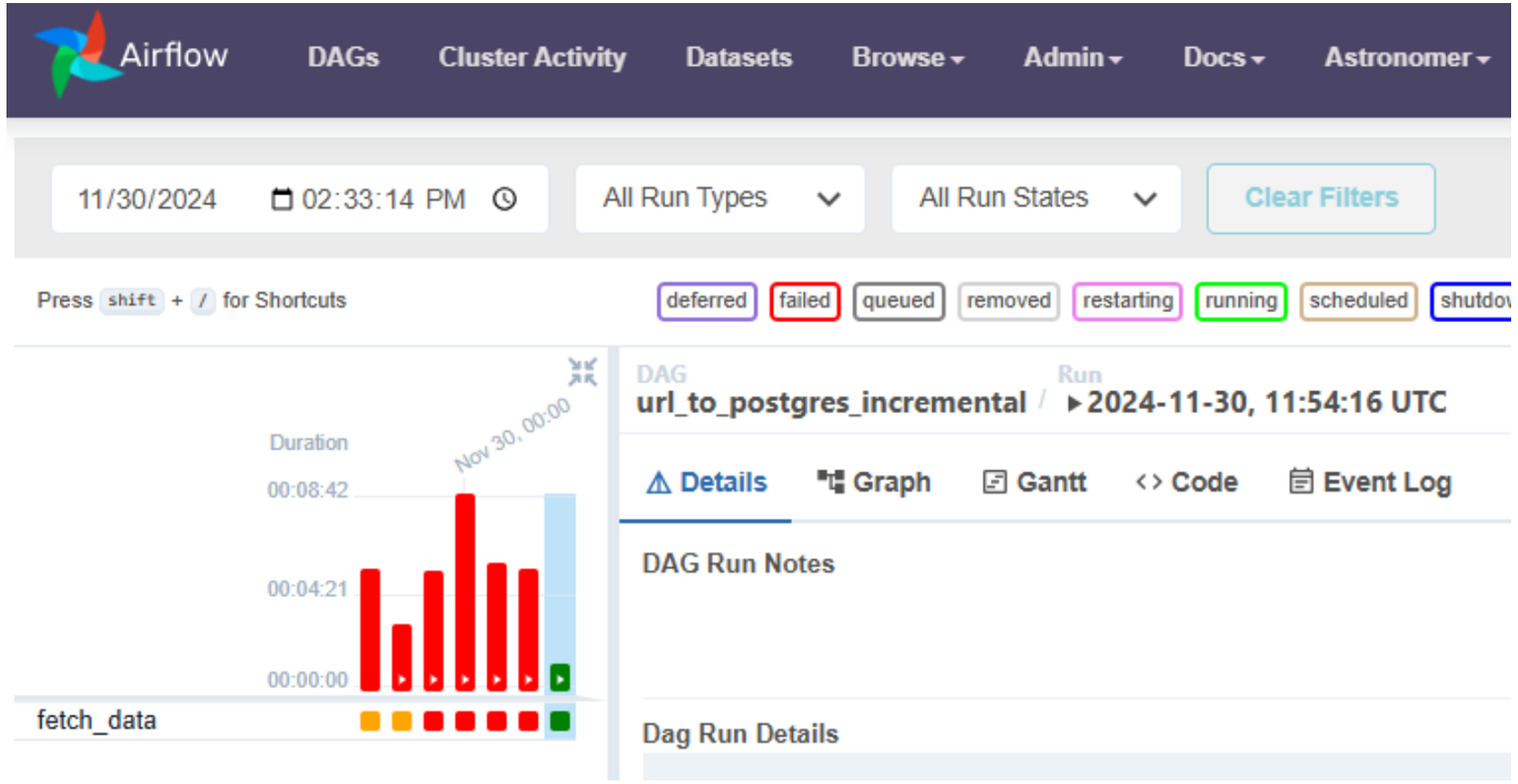
DAG (Directed Acyclic Graph): exemplo concreto

- DAGs proporcionam
  - baixo acoplamento e alta coesão
  - Encapsulmanento de complexidade
  - Rastreabilidade e auditabilidade



Task (Airflow)

- Uma instância de um Operator
  - Configuram parâmetros de contexto
    - ✓ Por exemplo, quando executar o Operator



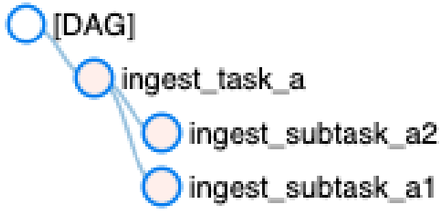
Dag Run

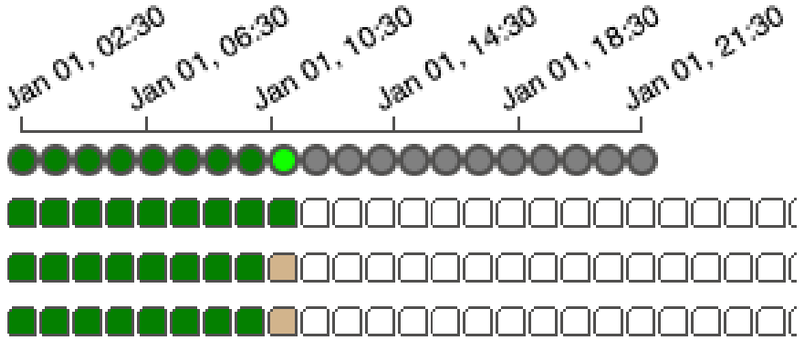
Task

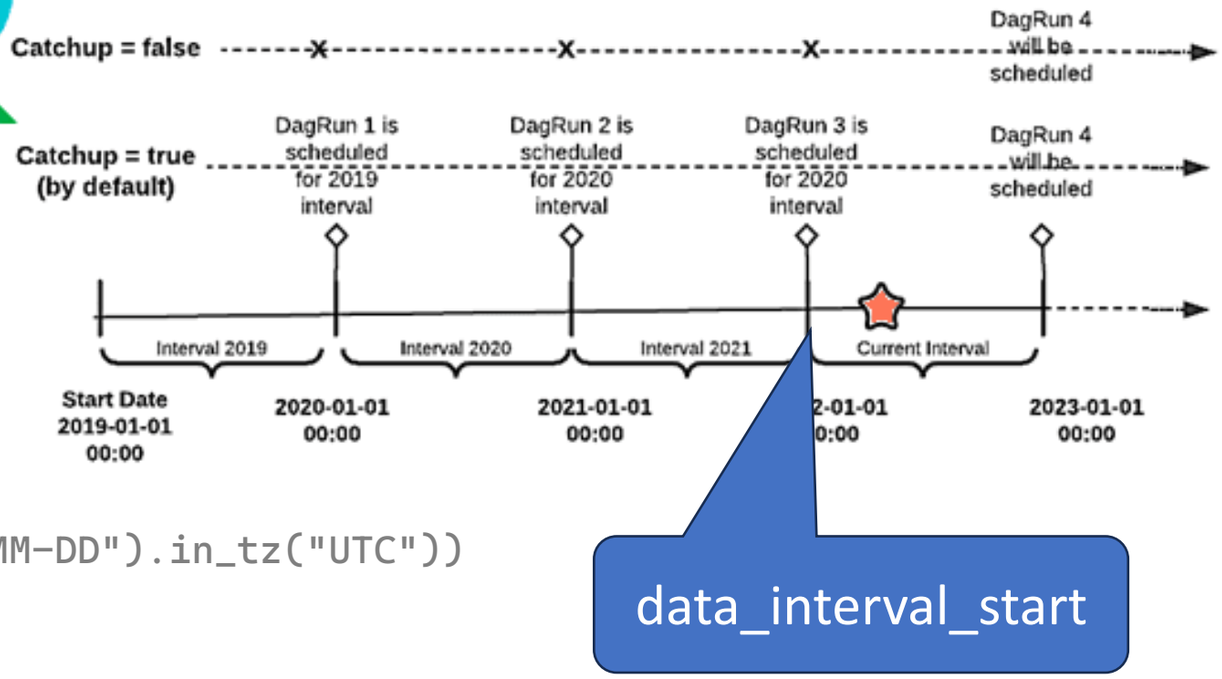
DAG simplificada

```
from airflow.decorators import dag
import pendulum

def python_1_func():
    print(3)

@dag(default_args={"retries": 0}, schedule="0 0 1 1 *",
    start_date=pendulum.from_format("2022-04-06", "YYYY-MM-DD").in_tz("UTC"),
    catchup=False
)
def Teste2():
    python_1_func()

dag_obj = Teste2()
```

DAG simplificada



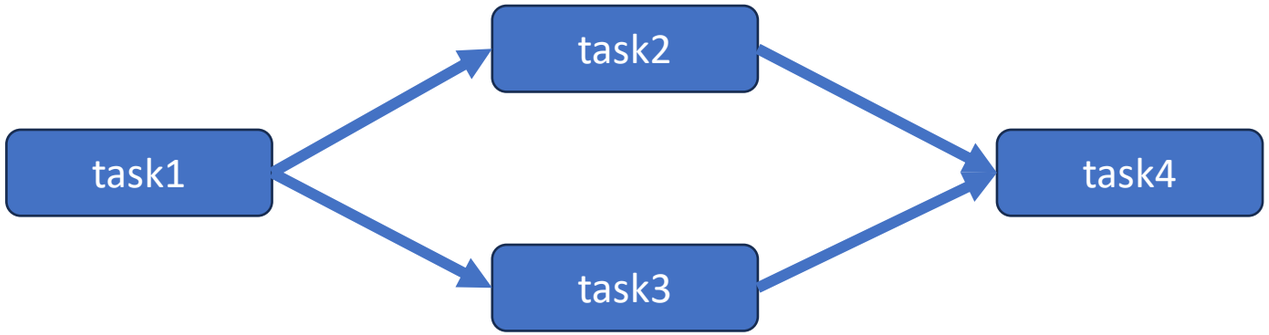





```
... (imports)

@task
def ingest_task_a(): print("Main ingest task A")

@task
def ingest_subtask_a1(): print("Subtask A1")

@task
def ingest_subtask_a2(): print("Subtask A2")

@dag(schedule="0 0 1 1 *", catchup=False,
    start_date=pendulum.from_format("2022-04-06", "YYYY-MM-DD").in_tz("UTC"))
def Teste2():
    a = ingest_task_a()
    a1 = ingest_subtask_a1()
    a2 = ingest_subtask_a2()
    a >> [a1, a2]  # Defines branching dependency

dag_obj = Teste2()
```

Tasks paralelas

```
task1 >> [task2, task3] >> task4
```



Agendamento de Execução

- start_date
  - Data a partir da qual a o agendamento periódico (CRON STRING)
    - ✓ Passa a valer
- CRON STRING
  - Usado para especificar o a recorrência do agendamento periódico
    - ✓ Ex.: todo dia, toda semana, todo mês, etc
- Catchup
  - Executa os agendamentos passados
    - ✓ Antes de hoje e depois de start_date
- data_interval_start
  - Data lógica que marca o início do intervalo de execução de uma DAG
- Retries
  - Quantidade de tentativas em caso de falha

Agendamento de Execução

```
@dag(
    dag_id='hello_world',
    start_date=datetime(2025, 9, 15),
    schedule='@daily',
    catchup=False,
    default_args={'retries': 1},
)
def hello_world_dag():
    hello_world_task = hello_world()
```

Agendamento com cron

“At 04:05.”

next at 2024-12-05 04:05:00 random

5 4 * * *

minute	hour	day (month)	month	day (week)
		*	any value	
		,	value list separator	
		-	range of values	
		/	step values	
@yearly			(non-standard)	
@annually			(non-standard)	
@monthly			(non-standard)	
@weekly			(non-standard)	
@daily			(non-standard)	
@hourly			(non-standard)	

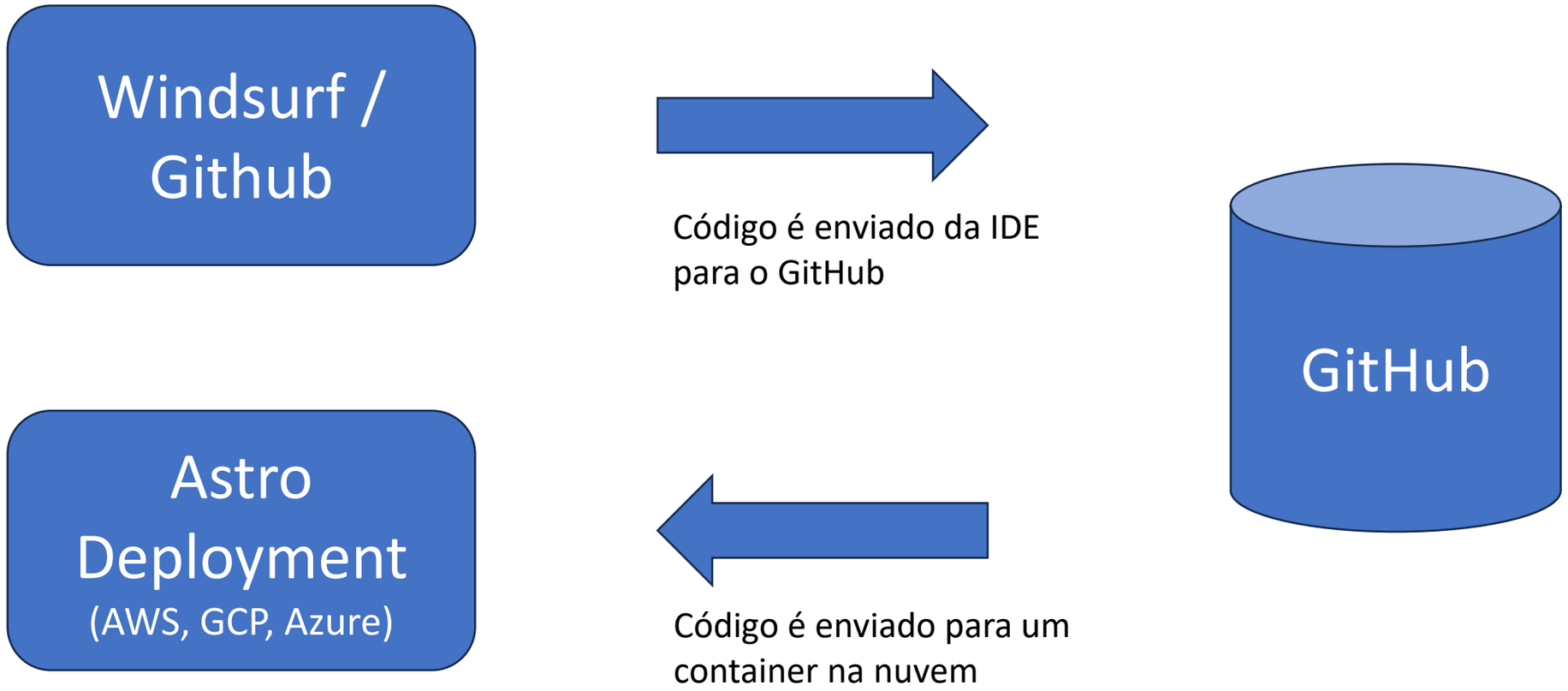
Airflow no Atronomer.io - Requisitos

- No astronomer.io
  - Workspace
  - Deployment
- No GitHub
  - Fork do repositório astro-dags-template
- Um banco de dados (ex. postgres na Digital Ocean)
- Acesso ao Google Looker Studio

astronomer.io

- Ferramenta online para gerenciar clusters do Airflow
- Trial de 14 dias
  - Ferramenta paga
    - ✓ Num contexto produtivo, o custo é efetivo

Deploy com IDE e GitHub



Atividade 9.1 (15 min)

- Fork do repositorio astro-dags-template
  - Repositório
    - ✓ https://github.com/alexlopespereira/astro-dags-template
  - Vídeo
- Criar uma conta no Astronomer.io
- Criar um Deployment
  - Vídeo
- Criar um Token de API
  - Vídeo

Atividade 9.2 (10 min)

- Configurar a Action no Github - Vídeo
  - Crie uma variavel e coloque o Deployment ID
  - Crie uma secret e coloque o Token da API
- Criar um commit para rodar um Workflow - Vídeo
  - Executar manualmente a dag hello_world no Airflow
    - ✓ E verificar os logs no Airflow

Atividade 9.3 (5 min)

- Configurar a Conexão com Postgres
  - Configurações de conexão
    - ✓ `username = doadmin`
    - ✓ `password = PASSADO PELO PROFESSOR`
    - ✓ `host = db-postgresql-nyc3-66496-do-user-3481063-0.k.db.ondigitalocean.com`
    - ✓ `port = 25060`
    - ✓ `database = defaultdb`
    - ✓ `sslmode = require`
      - `{"sslmode": "require"}`
  - Vídeo

Coleta de dados do preço do bitcoin (CoinGecko)

```
ctx = get_current_context()

# Janela "ontem": [data_interval_start - 1 dia, data_interval_start)
end_time = ctx["data_interval_start"]
start_time = end_time - timedelta(days=1)
# CoinGecko exige epoch em segundos (não ms)
start_s = int(start_time.timestamp())
end_s = int(end_time.timestamp())
url = "https://api.coingecko.com/api/v3/coins/bitcoin/market_chart/range"
params = {
    "vs_currency": "usd",
    "from": start_s,
    "to": end_s,
}

# Observação: CoinGecko pode aplicar rate limit (HTTP 429).
# O retry geral é tratado pelo Airflow (default_args['retries']).
r = requests.get(url, params=params, timeout=30)
r.raise_for_status()
payload = r.json()
```

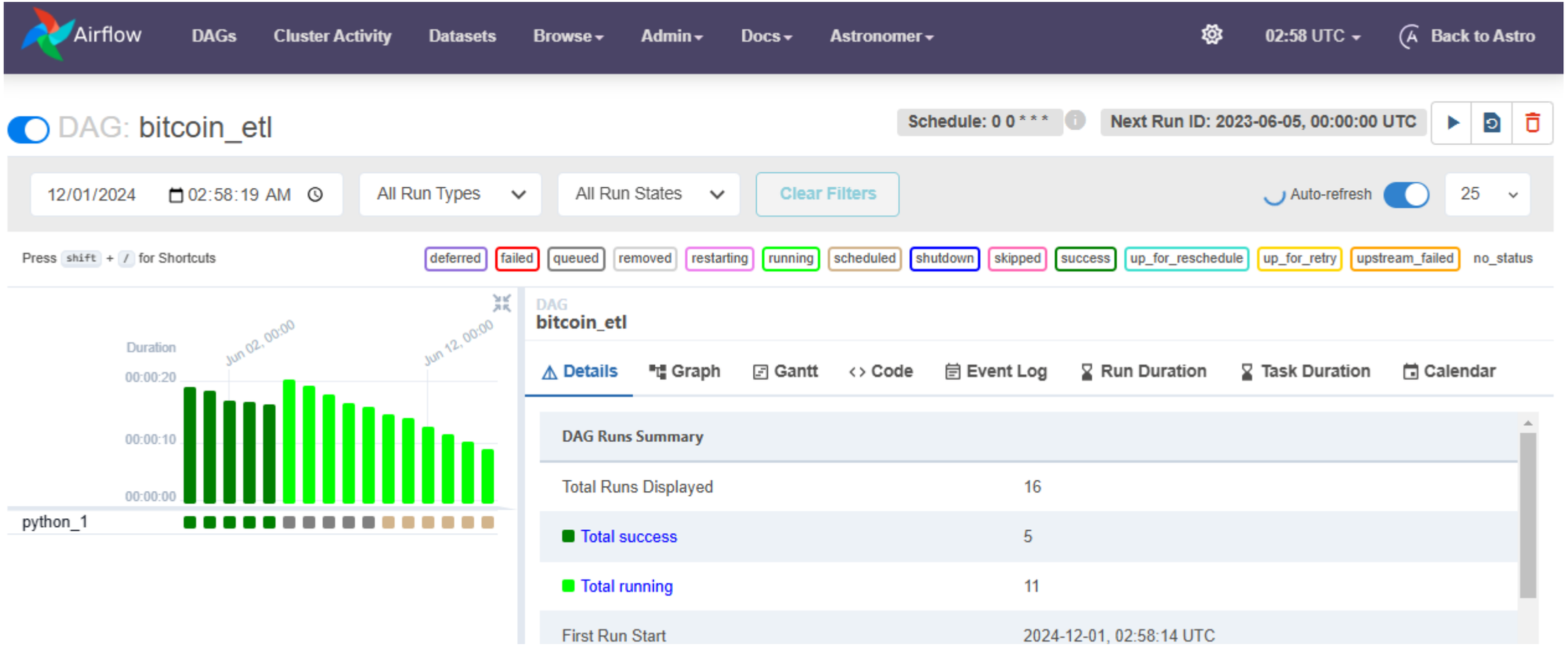
Coleta de dados do preço do bitcoin (CoinGecko)

```
# payload contém listas de pares [timestamp_ms, valor]
prices = payload.get("prices", [])
caps = payload.get("market_caps", [])
vols = payload.get("total_volumes", [])

if not prices:
    print("Sem dados retornados pela API para a janela especificada.")
    return

# Constrói DataFrames individuais
df_p = pd.DataFrame(prices, columns=["time_ms", "price_usd"])
df_c = pd.DataFrame(caps, columns=["time_ms", "market_cap_usd"])
df_v = pd.DataFrame(vols, columns=["time_ms", "volume_usd"])

# Merge por timestamp (ms)
df = df_p.merge(df_c, on="time_ms", how="outer").merge(df_v, on="time_ms", how="outer")

# Converte timestamp e organiza índice
df["time"] = pd.to_datetime(df["time_ms"], unit="ms", utc=True)
df.drop(columns=["time_ms"], inplace=True)
df.set_index("time", inplace=True)
df.sort_index(inplace=True)
```

Coleta de dados do preço do bitcoin (CoinGecko)

```
# Converte timestamp e organiza índice
df["time"] = pd.to_datetime(df["time_ms"], unit="ms", utc=True)
df.drop(columns=["time_ms"], inplace=True)
df.set_index("time", inplace=True)
df.sort_index(inplace=True)

# Preview no log
print(df.head(10).to_string())

# TODO: salvar no warehouse, ex. via PostgresHook / to_sql
from airflow.providers.postgres.hooks.postgres import PostgresHook
hook = PostgresHook(postgres_conn_id="postgres")
engine = hook.get_sqlalchemy_engine()
df.to_sql("bitcoin_history", con=engine, if_exists="append", index=True)
```

Coleta de 1 ano em andamento



Exercício 9.1

- Faça um ETL com Airflow
  - Das cotações (diário ou horário) do bitcoin de pelo menos 6 meses
    - ✓ Armazene os dados no banco de dados postgres disponibilizado pelo professor,
      - Utilize um sufixo "_MEU_NOME" para diferenciar das tabelas dos seus colegas
    - ✓ Ou no google bigquery
- Faça um Dashboard com os dados coletados no bitcoin
- Submeta aqui a URL do codigo fonte no github da sua DAG
  - Submeta no mesmo formulário o link público do seu dashboard

Cuidados e Boas Práticas

- Retries em ambiente de desenvolvimento
  - ZERO
    - ✓ IT makes things worse faster
      - With greate power, comes great responsibility
- ETL Incremental
  - Qual a chave de incremento?
    - ✓ updated_at, last_modified, versão, ou número sequential
      - Chaves estáveis / consistentes
  - Idempotência & deduplicação
    - ✓ Destino com upsert

Cuidados e Boas Práticas

- Confiabilidade operacional
  - Retries + backoff
    - ✓ defina retries e retry_delay sensatos.
  - SLAs e alertas
    - ✓ use SLAs e alertas (Slack)
  - Timeouts
    - ✓ execution_timeout por tarefa para evitar jobs presos.
  - Isolamento
    - ✓ max_active_runs=1 na DAG

Cuidados e Boas Práticas

- Timezone/clock skew
  - normalize para UTC; cuidado com origens com relógios desalinhados.
- Precisão do timestamp
  - milissegundos vs segundos
    - ✓ use >= + chave secundária para não perder empates.
- Mutabilidade
  - se registros antigos podem ser alterados, não use created_at; prefira updated_at.

Se entender é porque já está falando a língua dos nerds