

Data Factory

Padrão de Desenvolvimento

Histórico de versões

Versão	Data	Responsável	Descrição
1	28/07/2022	Ingrid de Oliveira Canaane - Ipiranga	Versão inicial do documento, com novos padrões estabelecidos pelo time de Arquitetura.

Sumário

1. Introdução	4
2. Branchs	4
3. Padrão para criação de objetos.....	4
3.1. Datasets.....	4
3.2. Pipelines	5
3.2.1. Atividades do pipeline	6
3.3. Dataflows	7
3.3.1. Steps internos do dataflow	7
3.4. Parâmetros	8
3.5. Triggers.....	8
3.6. Linked Services	9
3.7. Integration Runtimes	10
4. Organização.....	10
4.1. Datasets.....	10
4.2. Pipelines e Dataflows	12
5. Desenvolvimento de pipelines	14
5.1. Orientações Gerais	14
5.1.1. Ingestões com origem em bancos de dados	14
5.1.2. Arquivos de parâmetros dos pipelines.....	15
5.1.3. Criação de processos genéricos	16
5.1.4. Definição de agendamento de execução	16
5.1.5. Dados sensíveis	17
5.1.6. Utilização de Integration Runtime	17
5.1.7. Timeout de atividades de pipeline	18
5.1.8. Concorrência de pipelines	18
5.2. Ingestão de tabela dimensão – Oracle.....	19
5.3. Ingestão de tabela fato e agregada – Oracle	19
5.4. Ingestão de view – Oracle	23
5.5. Debugando pipelines e dataflows.....	24
6. Publicação de Alterações	25
6.1. Criação de Pull Request.....	25
6.1.1. Resolução de conflitos	27
6.2. Finalização do Pull Request.....	27

1. Introdução

O objetivo deste documento é instruir os desenvolvedores na construção dos procedimentos de ETL utilizando o serviço da Azure Data Factory, apontando os principais pontos de atenção quanto à organização, padronização e metodologia de trabalho, além de orientações para maior facilidade e agilidade no desenvolvimento.

2. Branchs

O Data Factory tem a opção de trabalhar com o modo GIT, onde é realizado o versionamento de cada alteração realizada. Sempre que um novo desenvolvimento for iniciado, deve-se criar uma nova branch, a partir da master, seguindo o padrão de nome abaixo:

1. Para projetos que usam Jira ou outras ferramentas que atribuam números/identificadores às atividades:
 - a. Padrão: [identificador da tarefa]_[breve descrição]
 - b. Exemplo: tarefa de identificador DLE-123 no Jira para ingestão da dimensão DM_PRODUTO, branch com nome *DLE_123_Ingestao_DM_PRODUTO*.
2. Para projetos que não caem no caso acima:
 - a. Padrão: [nome do projeto ou squad]_[breve descrição]
 - b. Exemplo: tarefa do projeto PE Regional para manutenção da ingestão da base de ORABI, PR_LANC_INDICADOR_GPLANFIN_MES, branch com nome *pe_regional_ajuste_tabela_fato*.

3. Padrão para criação de objetos

Para toda e qualquer criação de objetos no Data Factory, deve-se incluir em sua descrição comentários a respeito do objetivo e/ou funcionamento dele, conforme especificações feitas nesta seção. Além disso, os nomes de objetos devem seguir o padrão:

[prefixo]_[descrição abreviada].

As abreviações devem seguir o Dicionário de Abreviaturas disponível no [Portal Gestão de Dados](#). Com exceção dos steps internos do Dataflow e da branch, todos dos nomes de objetos devem ser escritos em letras **minúsculas**. Além disso, todas as atividades deverão ser parametrizadas para o **timeout** de até 1 dia.

3.1. Datasets

A orientação em relação a datasets é de que sempre sejam utilizados os que já existem, conforme documento [Padrão de Datasets](#). Novos datasets só devem ser criados se nenhum dos existentes atender a necessidade do processo, e, caso o dado

seja de uma origem ainda não mapeada por nenhum linked service existente, a criação deve ser solicitada ao time de Arquitetura, conforme especificado na seção destinada a linked services. Os nomes devem seguir o padrão abaixo:

[prefixo tipo dataset]_[usuário de banco]_[instância do banco] ou
[prefixo tipo dataset]_[descrição da parametrização] ou
[prefixo tipo dataset]_[nome da storage account]_[descrição da parametrização],
quando o dataset apontar para outro storage que não o produtivo, **stippdatalakedev;** ou
[prefixo tipo dataset]_[usuário]_[ambiente], se o dataset apontar para o Salesforce.

Prefixo	Tipo
azdtb_deltalake	Delta Lake
azsql	Banco SQL Server
binario	Arquivo Binário
csv	Arquivo CSV
json	Arquivo JSON
mysql	Banco MySQL
odbc	Cache (Abadi, Alphalinc)
ora	Banco Oracle
parquet	Arquivo Parquet
sf	Salesforce
syn	Azure Synapse
txt	Arquivo TXT
xlsx	Arquivo Excel

Tabela 1: Prefixos de datasets

Exemplos:

- azsql_datalake_bi
- csv_param_file_name
- json_param_file_system
- odbc_abadidl_cache_prd
- ora_dletl_ipjdep
- ora_dletl_orabi
- parquet_ippstgdevatena_param_file_name (parquet que aponta para outro storage)
- parquet_param_file_name
- sf_integralsalesfc_ipirangarede (ambiente produtivo do Salesforce)
- xlsx_param_file_range

3.2. Pipelines

Na criação de pipelines, deve-se seguir o seguinte padrão para nomenclatura:

pip_[nome da tabela], no caso de ingestões com **origem** em bancos de dados, ou
pip_azsql_[nome da tabela], no caso de processos de **carga em** tabelas no Azure SQL; ou

pip_[descrição sucinta], para outros processos

Todos os pipelines deverão conter, **obrigatoriamente**, em sua descrição:

- Nome do engenheiro responsável pelo desenvolvimento e consultoria a qual pertence;
- Área solicitante (squad, projeto ou BU);
- Explicação da lógica que está sendo utilizada e da regra de negócio. Se possível, deve-se incluir exemplos.

Exemplo 1:

Nome: pip_pr_pneg_componente_propneg

Descrição: Pipeline responsável pela cópia de dados da tabela PR_PNEG_COMPONENTE_PROPNEG em ORABI para o lake na camada raw.

Engenheiro: Robson Cesar Gomes Quintino

Squad solicitante: Site Location

Exemplo 2:

Nome: pip_interface_sitelocation_salesforce

Descrição: Pipeline para preparação dos dados gerados pela squad de Site Location, a cerca de volume potencial mensal para postos, para carga na base INTEGRSALESFCMV.CG_VOLUME_POTENCIAL_POSTO_MES, para uso em integração do Salesforce.

Engenheiro: Ingrid de Oliveira Canaane

Squad: Site Location

3.2.1. Atividades do pipeline

Para atividades internas a um pipeline, seguir o padrão:

[prefixo atividade]_[descrição sucinta]

Prefixo	Atividade
af	Azure Function
appvar	Append variable
cp	Copy Data
dl	Delete
dtb	Databricks
fail	Fail
fe	ForEach
ft	Filter
getmet	Get Metadata
if	If Condition
lk	Lookup
pip	Execute Pipeline
scpt	Script
setvar	Set variable
sp	Stored Procedure
st	Switch

until	Until
vl	Validation
web	Web
wh	WebHook
wt	Wait

Tabela 2: Prefixos de atividades do pipeline

Cada atividade deve ter um nome explicativo com uma descrição sucinta sobre seu funcionamento, que faça sentido.

Exemplo 1:

Nome da atividade: dtb_ntb_armazenagem_terceiros

Descrição: Faz chamada do notebook dtb_etl_armazenagem_terceiros, que faz carga dos dados de armazenagem em e para terceiros.

Exemplo 2:

Nome da atividade: cp_carga_orabi_transient

Descrição: atividade que realiza a cópia dos dados incrementais da tabela em ORABI para o storage, na camada transient

3.3. Dataflows

O padrão de nomenclatura para dataflows é o seguinte:

df_[nome da tabela] ou

df_[descrição do objetivo]

É necessário incluir a descrição do objetivo do fluxo criado, e especialmente para dataflows que possuam regras complexas, deve-se incluir uma explicação da lógica aplicada. Dessa forma, facilita o entendimento do processo em futuras necessidades de manutenção.

Exemplo:

Nome: df_ag_lancamento_conta

Descrição: Dataflow responsável pelo particionamento dos dados da carga incremental da tabela AG_LANCAMENTO_CONTA na camada raw, utilizando a coluna NO_AM, e gravação da última data de inclusão dos registros ingeridos no arquivo json para controle de carga.

3.3.1. Steps internos do dataflow

Para atividades internas de dataflows, deve-se incluir o prefixo do tipo de step, mas diferentemente de outros objetos, pode-se utilizar letras maiúsculas para definição do nome.

[prefixo step][descrição sucinta]

Prefixo	Step
agg	Aggregate
alt	Alter Row

dc	Derived Column
ex	Exists
fl	Flatten
ft	Filter
jn	Join
lk	Lookup
ps	Parse
pv	Pivot
rk	Rank
sk	Surrogate Key
sk	Sink
sl	Select
spl	Conditional Split
sr	Source
st	Sort
str	Stringify
un	Union
upv	Unpivot
wd	Window

Obs.: “New Branch” adiciona novo fluxo, o nome ficará o mesmo do step selecionado.

Tabela 3: Prefixos de steps do dataflow

Exemplos:

- aggCountReg
- altUpdateReg
- dcFormataCols
- exData
- flRegistros
- ftDataNull
- jnDmVenda
- lkDmVenda
- pvValoresVenda
- skIdVenda
- skJson
- slRenomeiaCols
- splMesAnterior
- srTransient
- stRegistros
- unDmPontoVenda
- unpValoresVenda
- wdRankRegistros

3.4. Parâmetros

Os parâmetros de pipelines e dataflows devem ser criados com base no padrão:

p_[nome do campo]

Deve-se evitar o uso de nomes genéricos, procurando nomear com termos que descrevam bem o significado do parâmetro. Por exemplo, ***p_data_ult_carga***, no lugar de ***p_data1***.

3.5. Triggers

A criação de triggers, assim como de datasets, só deve ser feita se nenhuma das existentes atender a necessidade do processo, e, nesse caso, o nome deve seguir o padrão abaixo:

***trg_[tipo]_[horário de execução] ou
trg_[tipo]_[descrição do agendamento]***

Tipo	Descrição
schedule	Agendamento feito por definição do horário e frequência de execução. Utilizar preferencialmente o horário de Brasília (UTC-3).
event	Triggers por evento

Tabela 4: Identificadores de tipos de triggers

Exemplos:

- `trg_schedule_8_am` – execução diária às 8h
- `trg_schedule_7_am_weekdays` – execução às 7h, apenas em dias úteis
- `trg_schedule_6_am_once_a_week_monday` – execução às 6h, às segundas-feiras
- `trg_event_stone` – disparada sempre que um arquivo é criado no diretório especificado

A descrição de uma trigger deve conter a especificação de seu horário e frequência de execução, e no caso de triggers por evento, descrição deste evento.

Exemplo:

- Nome: `trg_event_fieldcontrol`
- Descrição: Trigger responsável por identificar a criação de um arquivo na pasta `raw/dados_externos/field_control/` e iniciar o pipeline `pip_interface_fieldcontrol`. Estes arquivos serão disponibilizados pela function 3 vezes por dia.

3.6. Linked Services

Caso necessário, a criação de linked services deve ser solicitada ao time de Arquitetura, que seguirá o padrão de nomes abaixo:

***ls_[prefixo linked service]_[referência ou descrição] ou
ls_[prefixo linked service]_[usuário]_[instância do banco]***, para linked services que fazem conexão com bancos de dados; ou
ls_[prefixo linked service]_[instância do banco], para linked services que fazem conexão com bancos externos; ou
ls_[prefixo linked service]_[nome do workspace], para linked services que fazem conexão com workspaces do Databricks;
ls_[prefixo linked service]_[usuário]_[ambiente], se o dataset apontar para o Salesforce.

Prefixo	Linked Service
af	Azure Function
azsql	SQL Server
deltalake	Delta Lake
dl	Storage Gen2

dtb	Databricks
fs	File System
http	HTTP
mysql	MySQL
odbc	Cache
ora	Banco Oracle
sf	SalesForce
sftp	SFTP
syn	Azure Synapse

Tabela 5: Prefixos de linked services

Exemplos:

- ls_af_fontesexternas_abegas
- ls_dl_stippdatalakedev
- ls_dtb_ipp_dev
- ls_mysql_ipiranga_piloto_dt_clean
- ls_ora_dletl_orabi
- ls_sf_integrsalesfc_ipirangarede

A descrição de um linked service deve conter seu objetivo, usuário usado e instância de banco acessada, quando for aplicável.

3.7. Integration Runtimes

Não se deve fazer criação de integration runtimes, uma vez que já existem IRs para as necessidades comuns do Data Factory. Qualquer necessidade nesse sentido deve ser levada para o time de Arquitetura, que avaliará o pedido.

Integration runtimes são nomeadas conforme o nome da máquina virtual à qual faz referência, e informando em sua descrição o objetivo.

ir_[nome da vm]

Exemplo: ir-vmdatalakeir-dev

4. Organização

4.1. Datasets

É obrigatório que os datasets utilizados sejam organizados em pastas por tipos de datasets.

São considerados tipos de dataset:

- Arquivos: por tipo de arquivo, dentro da pasta “files” (parquet, json, csv, avro, binary, orc etc.)

▲	files	136
▶	csv	36
▶	json	7
▶	parquet	81
▶	xlsx	12

Figura 1: Exemplo de organização de datasets do tipo arquivo

- Bancos: separados por instância. Para bancos externos, deve-se seguir a mesma regra.
 - Oracle

▲	oracle	27
▶	jde	3
▶	orabi	6
▶	orabi1	6
▶	orabif	5
▶	orasn	7

Figura 2: Exemplo de organização de datasets do tipo Oracle

- Azure SQL

▲	azure_sql	11
▶	sql-azure-sql-datalake-dev	1
▶	sqlldb-datalake-bi	5
▶	sqlldb-datalake-dev-controle-carga	1
▶	sqlldb-datalake-dev-datascience	3
▶	sqlldb-datalake-dev-sistneg	1

Figura 3: Exemplo de organização de datasets do tipo Oracle

- Cache

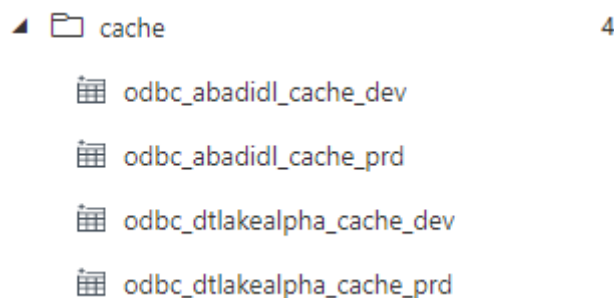


Figura 4: Exemplo de organização de datasets do tipo Cache

4.2. Pipelines e Dataflows

É obrigatório que pipelines e dataflows sejam organizados em pastas por assunto.

Regras:

1. Nomes de pastas e pipelines sempre em letras minúsculas e sem acentuação.
2. Estrutura a ser utilizada para organização dos pipelines:
 - 2.1. **Fontes internas:** pipelines/dataflows desenvolvidos para ingestão de dados internos à Ipiranga ou refinamentos específicos de projetos ou squads.
 - 2.1.1. **Oracle (e demais bancos):** ingestões de/para bancos de dados. A hierarquia de pastas deve ser seguida conforme instância do banco e owner da tabela. Pipelines que replicam views do Oracle também deverão seguir esse padrão, mesmo que não seja feita a cópia direta da view a partir do banco.


```
fontes_internas
  oracle
    [instância do banco]
      [owner da tabela]
        pip_[nome da tabela a]
        pip_[nome da tabela b]
  azure_sql
    [nome do banco]
      pip_azsql_[nome da tabela]
```
 - 2.1.2. **Projetos:** a pasta projetos compreende processos desenvolvidos por/para squads de Ciência de Dados e projetos. Deve ser organizado conforme o projeto/squad solicitante e o assunto.


```
fontes_internas
  projetos
    [nome do projeto ou squad]
      [assunto]
        pip_[descrição]
```
 - 2.1.3. **Views:** pipelines ou dataflows desenvolvidos para gerar visões a partir de dados existentes, mas não *especificamente* para uma squad ou projeto,

como processos mais genéricos de uso geral(avaliar com o time de Governança Técnica).

fontes_internas

views

pip_[descrição]

2.1.4. **SFTP:** pipelines ou dataflows desenvolvidos para ingestão via SFTP

fontes_internas

sftp

[assunto]

pip_[descrição]

▲	fontes_internas	538
	pip_corporativo	
▶	azure_as	2
▲	azure_sql	7
▶	sqlldb-datalake-bi_Copy	3
▶	sqlldb-datalake-dev-sistneg	4
▶	cache	2
▶	oracle	371
▶	projetos	150
▶	sftp	2
▶	views	3

Figura 5: Exemplo de organização de processos de “fontes_internas”

2.2. Fontes externas: pipelines e dataflows desenvolvidos para ingestão ou tratamento de dados externos à Ipiranga. Deve ser organizado conforme a área solicitante e fonte da informação.

fontes_externas

[nome do projeto ou squad]

[fonte de dado]

[assunto]

pip_[descrição]

▲	fontes_externas	42
	pip_fontes_externas	
▶	conecta	2
▶	contas_pagar	1
▶	contas_receber	1
▲	market_share	3
▶	datagro	1
▶	kpler	1
▶	line_up_derivados	1
▶	nps_consumidor	5
▶	previsao_demandas	26

Figura 6: Exemplo de organização de processos de “fontes_externas”

5. Desenvolvimento de pipelines

5.1. Orientações Gerais

5.1.1. Ingestões com origem em bancos de dados

Os datasets permitidos para uso em novos desenvolvimentos ou manutenções são os listados no documento [Data Factory - Datasets Padrão](#). A criação de novos datasets será permitida apenas em caso de necessidade, quando os já existentes não atenderem ao novo processo sendo construído, conforme especificado na seção 3.1.

Para os datasets referentes à bancos de dados, pode ser necessário solicitar acesso para a tabela a ser ingerida. Sendo assim, sempre que for iniciado o desenvolvimento de uma nova ingestão de banco de dados, a orientação é que seja testado, usando os datasets que constam no documento mencionado anteriormente, se é possível realizar um preview na tabela.

Caso a tentativa retorne um erro ORA-00942, é um indicativo de que:

- A tabela não existe na instância do banco informada; ou
- O usuário padrão do Data Lake não possui acesso para a tabela.

Exemplo: tentativa de leitura da tabela DMCONTR.AG_CONTRATO_CFP_PESSOA_AM, existente em ORABI1, mas para a qual o usuário do Data Lake não possui acesso.

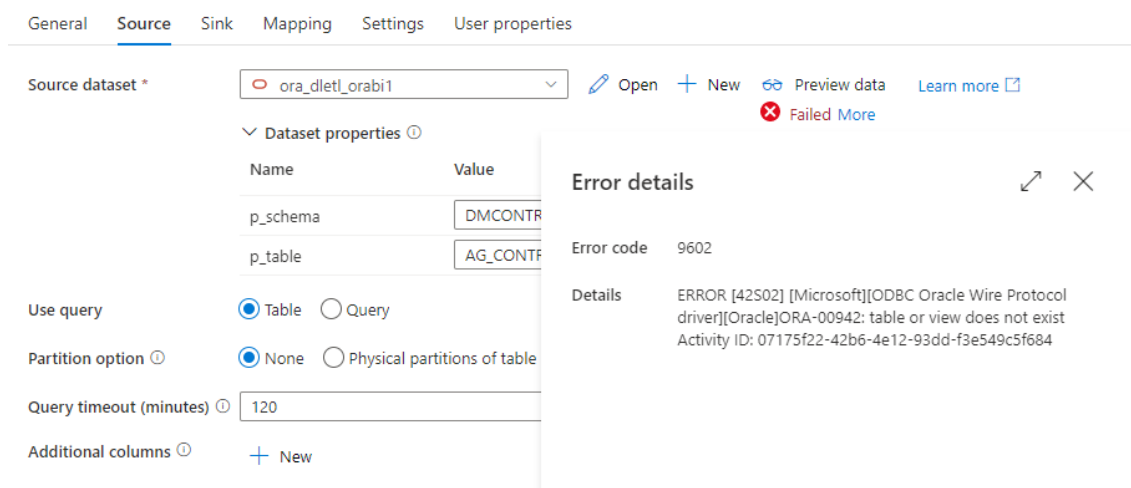


Figura 7: Exemplo de erro ORA-00942

Nesse caso, é necessário criar uma tarefa para o time de Administração de Dados (via ALM ou Jira), solicitando os acessos necessários para a tabela, para o usuário de banco:

- Oracle (ORASN, ORABI, ORABI1, ORABIF, JDE): usuário DLETL
- ABADI: usuário ABADIDL
- SQL Server (sqlldb-datalake-bi): usuários são definidos por projeto e o acesso às tabelas será concedido conforme necessidade de cada um (verificar com o time de AD ao abrir a solicitação).

5.1.2. Arquivos de parâmetros dos pipelines

Foi adotada a prática de utilização de parâmetros em pipelines e dataflows, para que em casos de manutenções, a alteração seja concentrada num único ponto. Algumas regras a serem seguidas:

- Os arquivos de parâmetros devem ser carregados na pasta ***"data/_conf/"***, em formato JSON;
- O nome de um arquivo deve sempre refletir o nome do pipeline a que pertence, sem o prefixo ***"pip_"***;
- É necessário incluir no JSON uma chave referente às informações do processo. Na descrição do processo, deve-se informar também o nome do engenheiro responsável pelo desenvolvimento, consultoria a que pertence e projeto atendido:

```
"processInformation": {
    "application": "DataFactory",
    "resourceName": "adf-ipp-datalake-dev",
    "applicationName": [nome do pipeline],
    "processDescription": [breve descrição do processo]
}
```

- Outras chaves do json são referentes às origens e destinos do processo, conforme exemplificado nas seções destinadas a ingestões, mas em suma, seguem o padrão abaixo:

```
"dataset[Destino ou Origem][Camada]": {
    "fileSystem": [container em que o dado será salvo ou lido],
    "directory": [camada e diretório em que o dado será salvo ou lido],
    "fileName": [nome do arquivo com extensão],
    "compression": [tipo de compressão, normalmente snappy]
}
```

- Os parâmetros devem ser passados pela opção **Add dynamic content**, exibida para cada parâmetro dos datasets. Ali são adicionados valores dinâmicos, que podem ser saídas de steps anteriores do pipeline ou funções que calcularão valores.

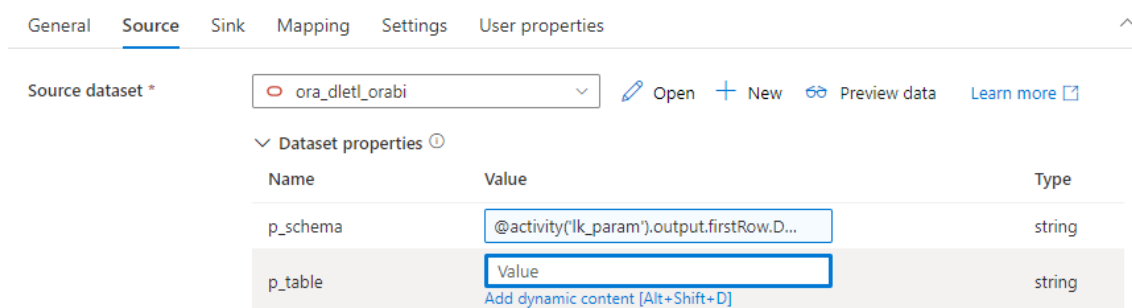


Figura 8: Exemplo de passagem de parâmetros para dataset

5.1.3. Criação de processos genéricos

É proibida a criação de processos genéricos para carga de bases de dados ou outros fluxos. Por processo genérico entende-se pipelines parametrizados de forma que atendam a mais de uma necessidade, passando apenas os parâmetros necessários para executá-lo.

Por exemplo, pipeline para carga de tabelas da instância ORABI, em que o owner e nome da tabela são parametrizados, e outros processos fazem chamada a esse pipeline passando os parâmetros para carga da tabela que desejam.

A orientação oficial é de que sejam criados pipelines individuais para cada novo processo, de forma que o monitoramento dessas cargas seja feito de forma mais fluida e facilmente identificável.

5.1.4. Definição de agendamento de execução

Para agendamento de processos específicos para squads ou projetos, o horário de execução deve ser alinhado com o demandante do desenvolvimento (analista de negócio, líder de projeto ou cientista de dados).

Para processos cuja origem são bancos de dados, deve-se verificar o horário de execução do processo de carga na origem para definir o melhor horário de carga do dado no Data Lake. Algumas formas de verificar:

- Pelas colunas de data de inclusão e/ou alteração (DT_INCL e DT_ALTER) das tabelas, buscando pela faixa de horário mais comum;
- Verificação do horário e frequência de agendamento da carga. Esta opção demanda uma consulta à equipe de sustentação, ao Power Center e/ou ao documento de procedimentos de carga para produção disponível no [Portal ASG](#).

5.1.5. Dados sensíveis

Quando for verificado que um dado sendo ingerido contém informação sensível, a ingestão do mesmo deve ser direcionada para um storage apartado, ao qual o acesso é concedido mediante alinhamento com justificativa da necessidade. Esse direcionamento é válido para qualquer tipo de base de dados, seja ela interna ou externa, vindo de sistemas ou sendo input de usuários.

O engenheiro responsável pelo desenvolvimento do processo deve sempre verificar o *schema* da base sendo ingerida, de forma a garantir que informações sensíveis sejam direcionadas para o local correto.

Algumas informações importantes sobre o storage de dados sensíveis:

- Storage account: stipdatalakelgpdhml
- Container: dados-sensíveis
- Estrutura de pastas: deve seguir o mesmo padrão do storage produtivo, mas o container usado é “dados-sensíveis”
- Caso haja necessidade de tratamento do dado via Databricks, deve ser solicitado acesso ao workspace destinado a este fim, ***dtb-ipp-datalake-dados-sensives***.

5.1.6. Utilização de Integration Runtime

Existe uma integration runtime otimizada que deve ser usada no desenvolvimento dos pipelines, ***AutoResolveIntegrationRuntimeTTLMem***. Ela deve ser selecionada na chamada de execução de dataflows, na aba ***Settings***, como na Figura abaixo.

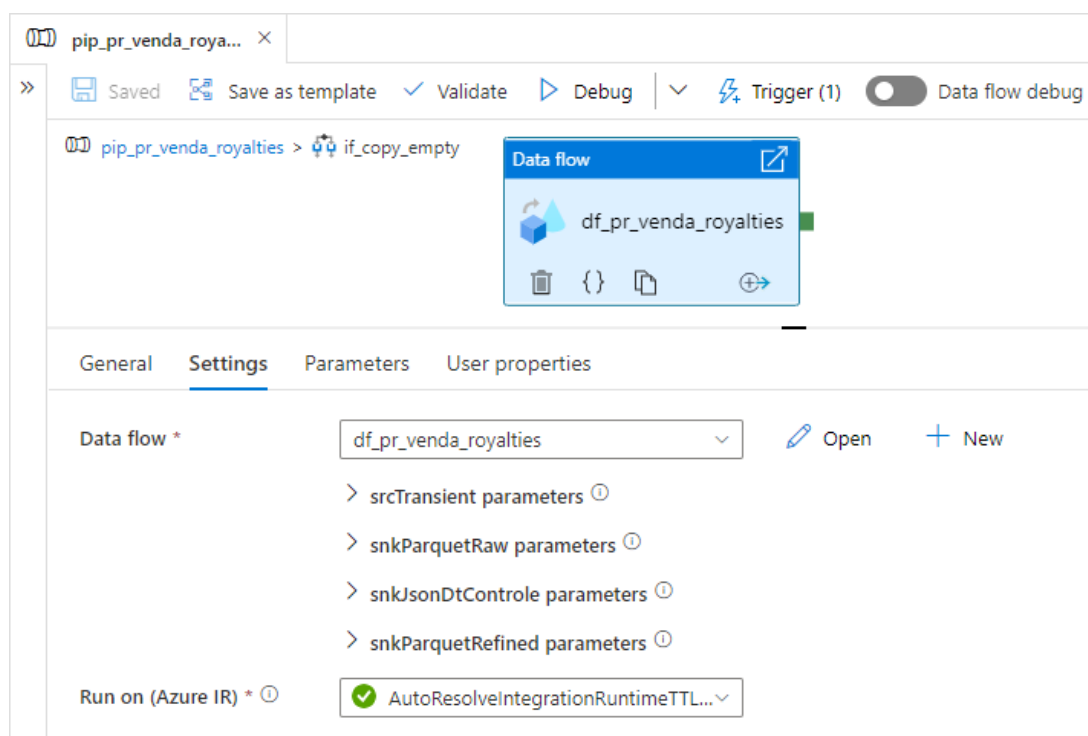


Figura 9: Exemplo de seleção de Integration Runtime para Data flow

5.1.7. Timeout de atividades de pipeline

Ao ser adicionada uma nova atividade a um pipeline, esta vem com um timeout padrão de 7 dias, porém, todas as atividades deverão ser parametrizadas para o timeout de até 1 dia (formato D.HH:MM:SS), na aba **General** da atividade em questão.

5.1.8. Concorrência de pipelines

Por padrão, não há um número máximo de execuções concorrentes que um pipeline pode ter. Porém, é interessante definir a concorrência para 1, de forma a evitar que, por qualquer motivo, múltiplas execuções de um mesmo pipeline se sobreponham, podendo causar erros. Dessa forma, outras execuções que tentem iniciar, serão colocadas em uma fila para aguardar o término da atual.

Nas configurações gerais do pipeline, na aba **Settings**, é possível definir esse número:

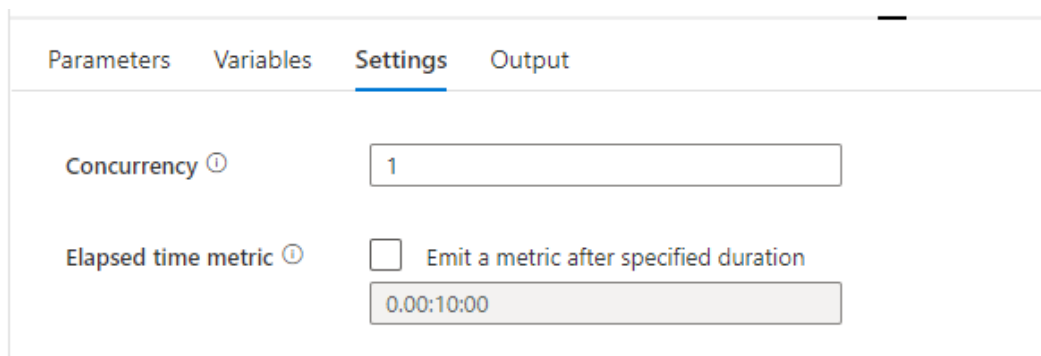


Figura 10: Exemplo de configuração de concorrência de pipeline

5.2. Ingestão de tabela dimensão – Oracle

Para tabelas dimensão, a carga deve ser feita de forma full, devido ao fato de o volume de dados dessas bases ser, em geral, baixo. Podem existir casos em que o volume de uma dimensão seja extremamente acima do comum, e para esses casos, é recomendado que a carga seja incremental, como será descrito na próxima seção.

Pipelines de ingestão de dimensões possuem tipicamente 3 atividades:

- Lookup para o arquivo de parâmetros;
- Atividade de cópia, com origem no Oracle e destino no storage, na camada transient;
- Atividade de cópia, com origem no Azure Synapse e destino no storage, na camada trusted.



Figura 11: Exemplo de pipeline de ingestão de dimensões

O objetivo da passagem do dado pelo Synapse é converter os tipos de dados de algumas colunas, que são trocados automaticamente pelo Data Factory para origens Oracle, por exemplo, inteiros sendo interpretados como *double*. Para isso, é passada uma query com um **cast** das colunas necessárias para o tipo correto. O pipeline **pip_dm_tipo_projeto_synapse** pode ser usado como referência.

O step referente ao Synapse é necessário apenas quando a dimensão em questão se enquadra no caso mencionado acima. Se as colunas da tabela são todas do tipo *varchar*, por exemplo, não ocorre o problema de troca de tipos, e, portanto, basta a atividade de cópia do Oracle diretamente para a camada trusted, sem passar o dado pelo Synapse.

5.3. Ingestão de tabela fato e agregada – Oracle

Tabelas fato e agregadas são naturalmente maiores em volume de dados, de forma que realizar carga full delas se torna inviável. Portanto, para esses casos desenvolvemos processos de carga incremental, que verificará, a cada execução, os registros que foram incluídos no banco de dados desde a última carga para o Data Lake.

Normalmente é necessário realizar uma carga histórica inicial, trazendo os dados dos últimos x meses ou anos para as tabelas que serão usadas num novo desenvolvimento. Esse período de dados deve ser alinhado entre o engenheiro e o demandante (analista de negócio, líder de projeto ou cientista de dados). Não é necessário buscar todo o histórico de uma tabela, a menos que seja solicitado (e nesse caso, o histórico deve ser trazido aos poucos para o storage, por mês ou ano a depender da volumetria, e particionado no mesmo padrão usado para a carga incremental da tabela).

Diferente do que ocorre para dimensões, os pipelines de tabelas fato e agregadas possuem mais atividades:

- Lookup para o arquivo de parâmetros;
- Lookup para o arquivo de controle de carga, que contém a data a ser utilizada para trazer os dados;
- Atividade de cópia, com origem no Oracle e destino no storage, na camada transient;
- Condição, que verificará se a atividade de cópia trouxe algum dado;
- Dataflow, para particionamento dos dados;
- Atividade de deleção para o arquivo inicial.

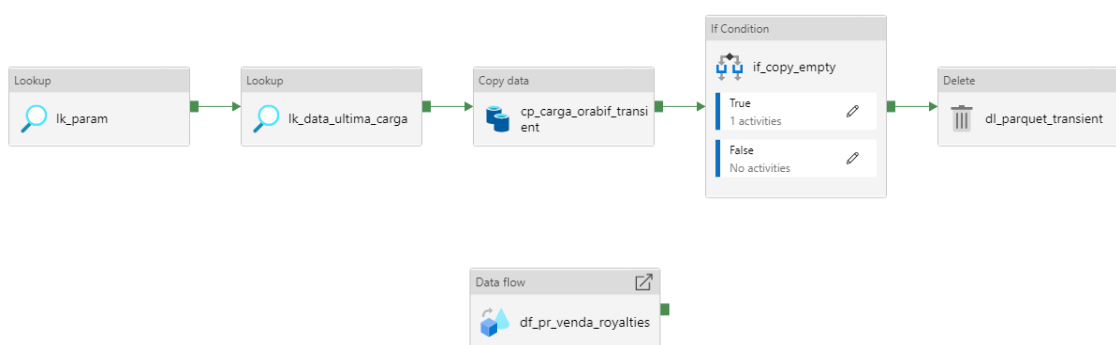


Figura 12: Exemplo de pipeline de ingestão de tabelas fato

Iniciando pelo arquivo de parâmetros, além da primeira chave com os dados sobre o processo, mencionada na seção 5.1.2, ele deve conter algumas outras:

- Origem do dado

```
"datasetOrigem": {
  "srcScheme": "[owner da tabela no banco]",
  "srcTable": "[nome da tabela]",
  "srcQuery": "[SQL para consulta dos dados na origem, considerando incremental]"
}
```
- Destino temporário, camada transient

```
"datasetDestinoTransient": {
  "fileSystem": [container em que o dado será salvo],
  "directory": [camada e diretório em que o dado será salvo],
  "fileName": [nome do arquivo com extensão],
  "compression": [tipo de compressão, normalmente snappy]
}
```
- Destino final, camada raw ou trusted

```
"datasetDestino": {
  "fileSystem": [container em que o dado será salvo],
  "directory": [camada e diretório em que o dado será salvo],
  "fileName": [nome do arquivo com extensão],
}
```

```
"compression": [tipo de compressão, normalmente snappy]
}
▪ Informações para controle de carga
  "datasetControleCarga":{
    "dtControleCarga": [coluna que será usada para controle da carga
    incremental, normalmente DT_INCL],
    "dtControle": [coluna de data que será usada para particionamento do
    dado],
    "jsonFileSystem": [container em que JSON de controle será salvo],
    "jsonDirectory": [diretório onde o JSON será salvo, padrão:
    "_conf/controle_data"],
    "jsonFileName": [nome do arquivo JSON, padrão: dt_controle_[nome da
    tabela].json]
  }
```

A verificação de novos registros é tipicamente feita pela data de inclusão e/ou alteração, como no exemplo a seguir. A query busca todos os registros da tabela que foram incluídos ou alterados desde a data % (caractere substituído pelo valor recuperado na segunda lookup, do arquivo de controle de carga), até a data \$ (caractere substituído pela data do início de execução do processo).

```
select *
  from DMFRANQ.PR_VENDA_ROYALTIES
 where (trunc(DT_INCL) > to_date('%', 'YYYY-MM-DD') and
        trunc(DT_INCL) < to_date('$', 'YYYY-MM-DD'))
        or (trunc(DT_ALTER) > to_date('%', 'YYYY-MM-DD') and
            trunc(DT_ALTER) < to_date('$', 'YYYY-MM-DD'))
```

A query usada pode ser adaptada ou reformulada conforme necessidade do processo, de acordo com a lógica da carga dos dados na origem. Para isso, é importante fazer uma análise do processo de carga sempre que possível, de forma a entender a periodicidade com que os dados são inseridos ou atualizados, como são atualizados, se existe reprocessamento e outras questões semelhantes.

Dito isso, o arquivo inicial será gravado na camada transient, e a atividade de condição verificará se o número de linhas que a atividade de cópia retornou é maior que zero. Isso é necessário pois a tabela pode não ter sido atualizada na origem até o momento da carga. Em caso positivo (*True*) o fluxo passa por um dataflow interno à condição. Em caso negativo, segue adiante e deleta o arquivo da camada transient.

A seguir, os dados trazidos serão lidos por um dataflow que realizará o particionamento desses dados na camada seguinte, raw (ou trusted, caso a lógica de carga usada contemple reprocessamento e não gere duplicidade de registros). O particionamento é feito por data e, para isso, é necessário definir a coluna que será usada.

Sempre que possível, recomenda-se particionar o dado pela data de referência da tabela em questão (por exemplo, DT_REF, DT_TRANS, NO_AM), o que permite maior

transparência na organização dos dados e consequente otimização da leitura dos mesmos por processos. Quando não, o particionamento é feito pela data de inclusão/alteração dos registros. Nesse caso, se o registro possuir data de inclusão e alteração preenchidas, deve ser considerado para o controle a data mais recente.

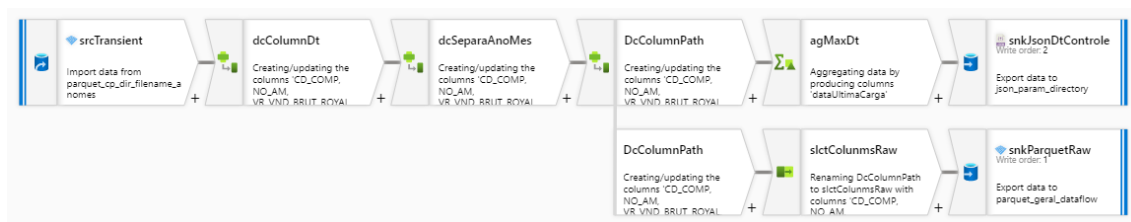


Figura 13: Exemplo de steps de um Data flow

Após a leitura da transient, o segundo e terceiro steps criarão as colunas que apoiarão o particionamento do dado. Serão criadas as colunas de ano, mês e dia, com base na data de referência escolhida para a carga (caso essa data seja no nível mês, a coluna de dia não se faz necessária). Além disso, essa atividade de *Derived Column* pode ser usada para converter os tipos de dados de outras colunas da base.

O quarto step é onde se define a “partição” onde cada registro será salvo, com base numa coluna string, de nome “path” por exemplo, que é montada a partir da concatenação de alguns parâmetros passados para o dataflow, com as colunas de data definidas nos steps anteriores:

```
$p_camada + $p_directory+'/' + ano+'/' + mes+'/' + replace($p_file_name, '%', (ano+mes))
```

Os três parâmetros se referem a, respectivamente, camada, diretório e nome do arquivo do Data Lake onde o dado será gravado, seguindo os padrões de camadas definidos. No exemplo usado aqui, para um registro cuja coluna NO_AM possui o valor “202111”, a string ficaria da seguinte forma:

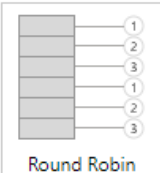
```
raw/dados_internos/bi/dmfranq/pr_venda_royalties/2021/11/rw_pr_venda_royalties_202111.parquet
```

A seguir, o fluxo se divide em dois. O primeiro passará por um *select*, que irá manter apenas as colunas originais da tabela sendo ingerida, mais a coluna “path” criada. Nesse step, é importante selecionar, na aba **Optimize**, a opção de particionamento por chave, usando a coluna path.

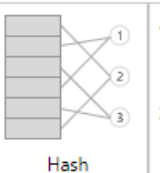
Select settings **Optimize** Inspect Data preview

Partition option * ☐ Use current partitioning ☐ Single partition ☒ Set partitioning

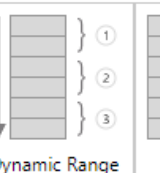
Partition type *



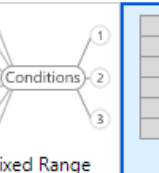
Round Robin



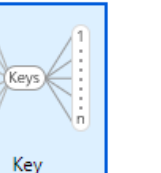
Hash



Dynamic Range



Fixed Range



Key

Unique value per partition *

Key column

abc path
+
🗑️

Figura 14: Exemplo de configuração de particionamento por chave

Passa-se então para o step final, o **Sink**, onde o dado é finalmente gravado. É aqui onde o particionamento é realmente definido. Na aba Settings, a opção **File name option** permite que o diretório dos dados seja definido com base em uma coluna. Dessa forma, cada registro será direcionado para um arquivo, de acordo com a string *path* que foi montada para ele.

Sink **Settings** Mapping Optimize Inspect Data preview

Clear the folder ☐

File name option * Name file as column data

Column data ⓘ abc path

Figura 15: Exemplo de configuração do particionamento dos dados do Data flow

O segundo fluxo é onde será definida a data de controle para a próxima carga. Uma atividade de agregação obterá a maior data de inclusão (DT_INCL) existente nos registros, e em seguida, gravará esse valor em um arquivo JSON que será lido pela segunda lookup do início do processo.

Pipelines que podem ser usados como referência:

- Informação nível ano mês: *pip_pr_venda_royalties*;
- Informação nível dia, tabela possui colunas de data de inclusão e alteração: *pip_pr_pesquisa_prc_bomba*.

5.4. Ingestão de view – Oracle

Para o caso de views existentes nas bases de dados da Ipiranga, a orientação é que não seja feita ingestão desses objetos, para evitar a oneração do ambiente. Quando houver necessidade de uso de uma view para desenvolvimento de um processo, deve-

se fazer ingestão das bases envolvidas no DDL e reproduzir a query no processo que a utilizará.

A depender da view, pode ser construído um processo via Databricks, que consumirá do storage os dados das tabelas envolvidas na view, reproduzirá o DDL e salvará o arquivo resultante no storage (seguindo o mesmo padrão de diretórios usado para tabelas do Oracle).

5.5. Debugando pipelines e dataflows

Para testar se as saídas de cada atividade estão conforme o esperado, usamos o recurso de **Debug** do Data Factory.

Começando pelo dataflow, para visualizar os outputs intermediários dos steps internos, usamos o **Data Preview** de cada um deles. Mas para isso, precisamos iniciar o **Data flow debug**:

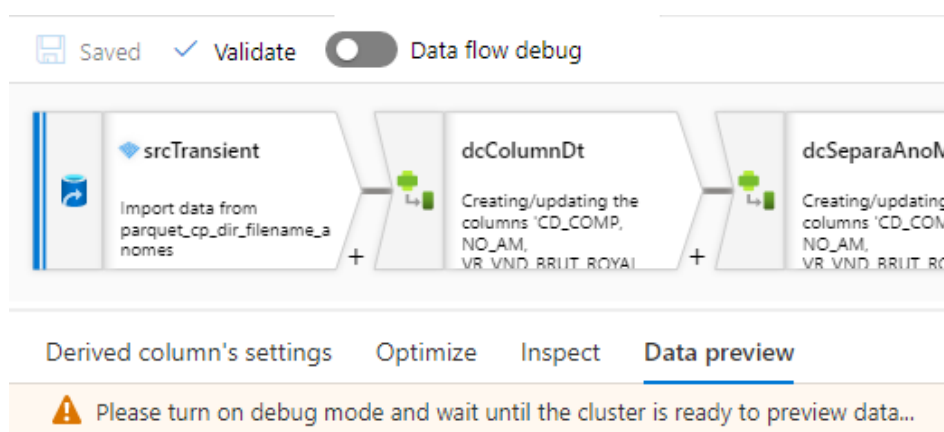


Figura 16: Exemplo de ativação do Data flow debug

Com ele ligado, é necessário informar o conteúdo de cada parâmetro usado pelo dataflow, em **Debug Settings > Parameters**. Ali, os parâmetros devem ser preenchidos exatamente como estão no arquivo JSON do processo. Feito isso, basta dar um **Refresh** em **Data Preview**.

Importante mencionar que o debug de um dataflow apenas gera previews dos dados, não realiza qualquer alteração no storage. Já o debug de um pipeline faz uma execução real, e, portanto, deve ser usado com cautela.

Ainda sobre pipelines, é possível também debugar até certo ponto do fluxo, com a opção **Debug until**. Ao clicar em uma atividade do pipeline, aparecerá no canto superior direito dela um círculo vermelho não preenchido, que indica que a opção de **debug until** está desabilitada.

Por exemplo, caso o desenvolvedor queira testar apenas as saídas dos arquivos de parâmetros, basta selecionar a última lookup e habilitar a opção. O círculo vermelho ficará preenchido, e as atividades seguintes ficarão “apagadas”:

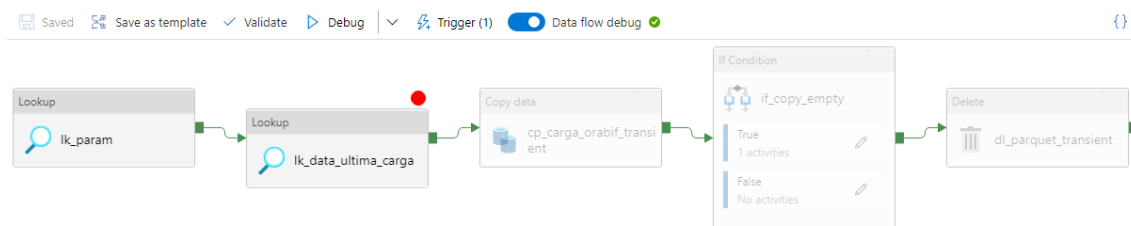


Figura 17: Exemplo de uso do “Debug until”

Feito isso, basta clicar em Debug para iniciar a execução do trecho do pipeline habilitado. O resultado pode ser visualizado na aba Output, onde o ícone exibirá a entrada recebida pela atividade de lookup, e o ícone exibirá a saída, que, no nosso exemplo, será o conteúdo do arquivo JSON.

Parameters Variables Settings Output							
Pipeline run ID: f420e43f-f125-4def-897c-c041fba84cbe View debug run consumption							
Name	Type	Run start	Duration	Status	Lineage status	Integration runtime	Run ID
lk_data_ultima_carga	Lookup	2021-11-26T19:41:05.655	00:00:04	Succeeded		DefaultIntegrationRuntime (East US 2)	451668ea-35
lk_param	Lookup	2021-11-26T19:41:00.399	00:00:04	Succeeded		DefaultIntegrationRuntime (East US 2)	ced3621b-9a

Figura 18: Exemplo de output do pipeline

6. Publicação de Alterações

Terminado o desenvolvimento, as alterações devem ser publicadas, seguindo alguns passos e requisitos descritos nesta seção.

6.1. Criação de Pull Request

Para publicar as alterações feitas é necessário criar um Pull Request. Antes de qualquer coisa, é recomendado validar a branch pelo próprio Data Factory, usando a opção **Validate all**, que retornará se a branch está ok, ou se possui qualquer inconsistência, indicando, nesse caso, onde estão os erros.



Figura 19: Exemplo de onde acionar a opção “Validate all”

Validada a branch, pode-se criar o pull request para seguir com o deploy das alterações. Com a branch em questão aberta, basta clicar na seta ao lado do nome da branch e ir até a opção **Create pull request**, que o direcionará para uma página do DevOps para inserir as informações a respeito do pull request.

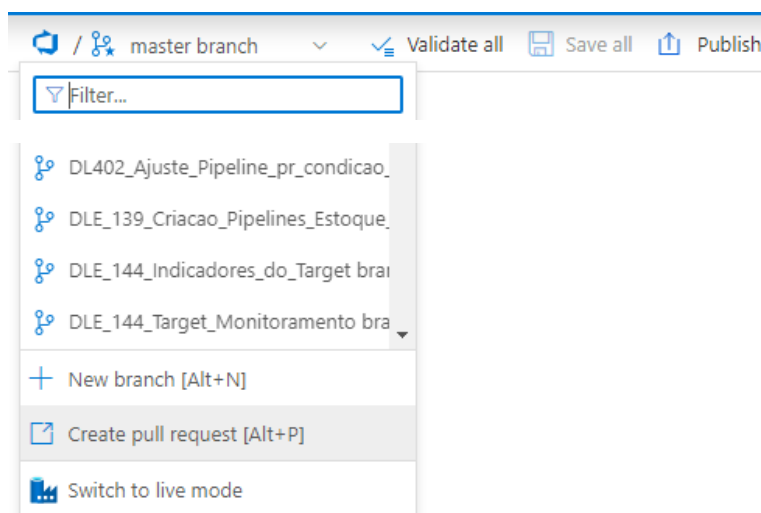


Figura 20: Exemplo de como criar Pull Request a partir do ADF

Nessa página, é mandatório preencher os comentários para o versionamento, identificando o objetivo da alteração sendo feita, o número da tarefa que gerou a manutenção/desenvolvimento e o projeto a que está atendendo.

Projeto: nome do projeto ou squad

Tarefa: identificador da tarefa, se houver

Descrição: descrição da demanda, explicando o objetivo de novos desenvolvimentos ou motivo das alterações, listando os objetos e/ou processos afetados.

Evidências de execução: incluir evidências de execução e validação da branch, com prints dos itens: execução dos pipelines (aba output); arquivos gerados no storage; Factory Validation Output (com data e hora).

Exemplo:

Squad: Pricing ME

Tarefa: DLE-1026

Descrição:

Foi construído o pipeline "pip_ap_parecer_acao_preco" , uma vez que para replicar o relatório de Margem Canceladas se faz necessário o uso das informações contidas nesta tabela. O pipeline está vinculado na trigger "trg_schedule_7_30_am".

Diretório no Data Factory:

fontes_internas/oracle/orasn/apco/

Diretório no Data Lake:

raw/dados_internos/sistema_de_negocio/apco/ap_parecer_acao_preco/yyyy/mm/dd

E sobre o pipeline "pip_margem_em_fluxo", foi adicionado o passo "dl_parquet_refined" para deletar os parquets que já estavam no diretório antes da inserção de novos dados.

6.1.1. Resolução de conflitos

Se a criação do pull request indicar existência de conflitos entre a branch de origem e a master, tais conflitos devem ser analisados. Em geral, as alterações já existentes na master devem ser sempre mantidas. Quando um conflito for relacionado a triggers, é provável que duas branches tenham sido criadas aproximadamente na mesma época, ambas incluíram pipelines numa mesma trigger, mas uma foi produtizada antes. Nesse caso, mantém-se a versão da master e, depois de finalizado o pull request, cria-se uma nova branch e pull request apenas para inclusão da trigger. Casos mais complexos devem ser analisados junto a equipe de Governança Técnica.

6.2. Finalização do Pull Request

Algun membro do grupo de aprovação poderá questionar algum ponto ou solicitar ajustes no pull request, para adequar aos padrões definidos. Após aprovado o PR, deverá ser concluído pelo engenheiro que o criou.

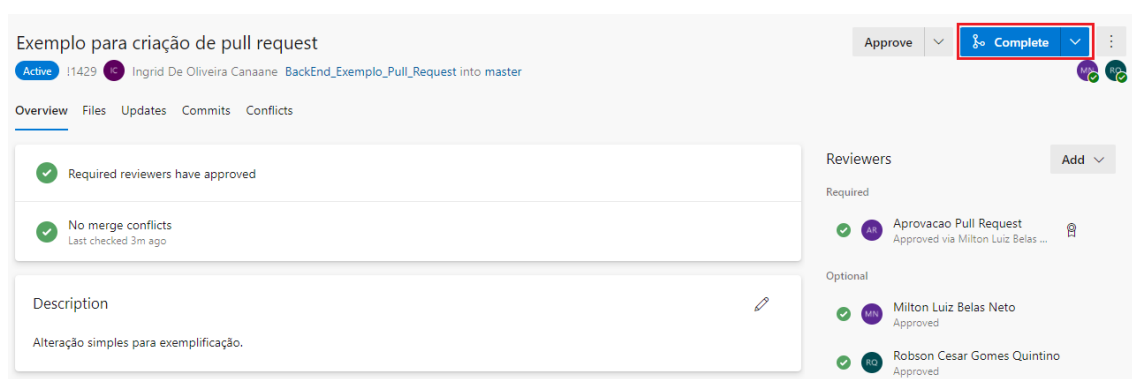


Figura 21: Exemplo de como completar um Pull Request

Ao completar o merge entre as branches, é orientado que a branch de origem seja excluída.

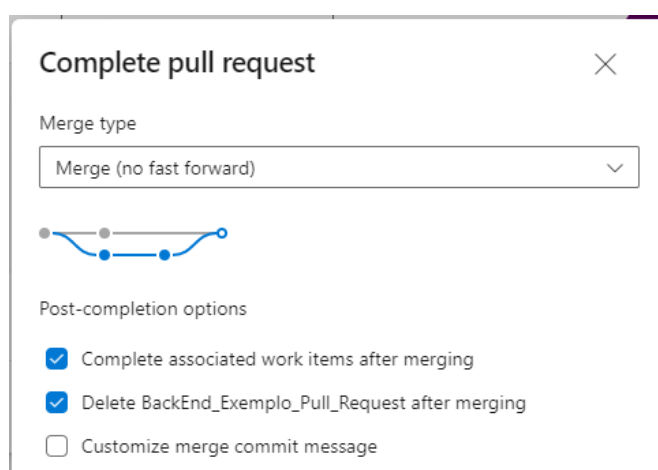


Figura 22: Exemplo de onde marcar a opção de deletar a branch após finalização do Pull request

Completada a etapa realizada no Azure DevOps, deve-se retornar ao Data Factory para publicar o que foi feito. Recomenda-se novamente realizar a validação da branches, pelo **Validate all**, e então publicar, pela opção **Publish** (exibida na Figura 19). As alterações serão listadas e o desenvolvedor deverá dar o Ok para a publicação iniciar.

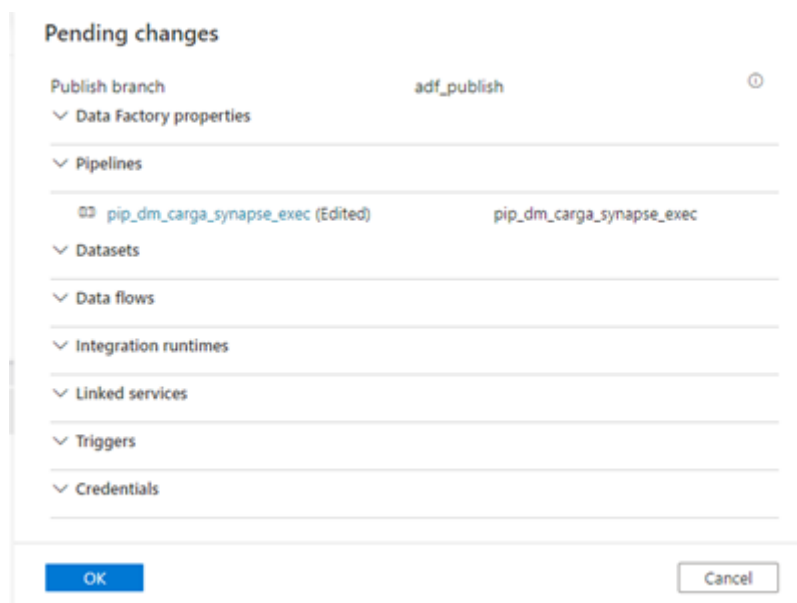


Figura 23: Exemplo de tela de publicação de alterações no ADF

Após o **Ok**, no ícone de notificação poderá ser conferido se a publicação e a geração de ARM foram concluídas com sucesso.

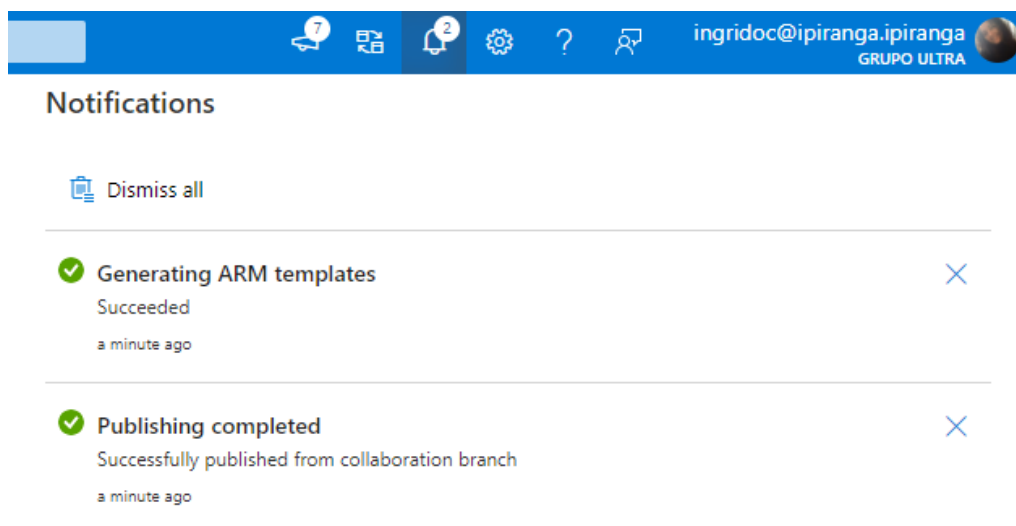


Figura 24: Exemplo de notificação de publicação com sucesso

Ocorrendo algum erro na publicação das alterações, deve-se verificar os detalhes para entender que ação deve ser tomada. Caso essas alterações contenham deleção de um pipeline, por exemplo, um erro associado a **Lock** do recurso será retornado (Figura abaixo). Nesses casos, a orientação é procurar a equipe de Governança Técnica para que o lock seja removido e a publicação concluída.

Error



Error code: OK
Inner error code: ScopeLocked
Message: The scope '/subscriptions/7deebbf6-cf9c-45b4-a02b-789e95a561b1/resourcegroups/rg-ipp-datalake-dev/providers/Microsoft.DataFactory/factories/adf-ipp-datalake-dev/triggers/trg_schedule_16h' cannot perform delete operation because following scope(s) are locked: '/subscriptions/7deebbf6-cf9c-45b4-a02b-789e95a561b1/resourceGroups/rg-ipp-datalake-dev/providers/Microsoft.DataFactory/factories/adf-ipp-datalake-dev'. Please remove the lock and try again.

Figura 25: Exemplo de erro por Lock