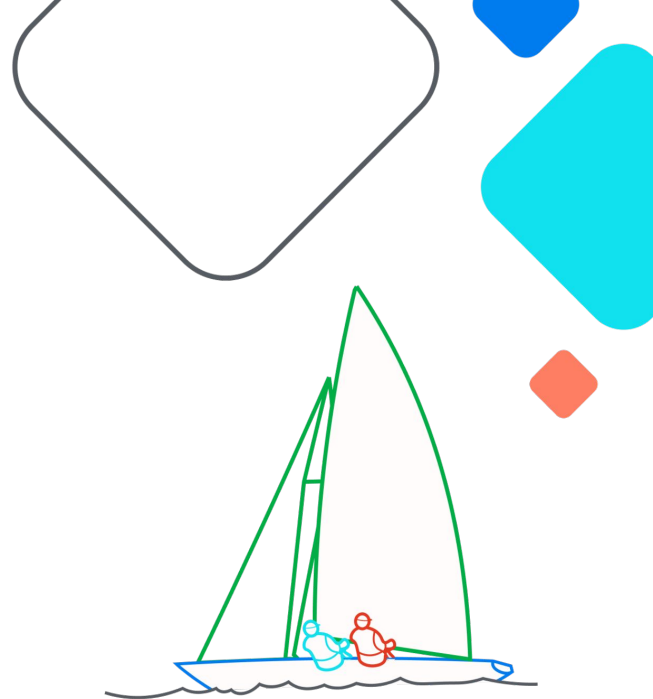


Accelerating Data Delivery

How the FT automated its
ETL pipelines with Airflow

Zdravko Hvarlingov & Vladi Nekolov



Airflow Summit
Let's flow together

September 19-21, 2023,
Toronto, Canada

Big Data in FT

- Important decisions taken based on data
- More and more data use cases across the company
- **Our legacy batching solution was not up for the job**

Airflow was the perfect fit

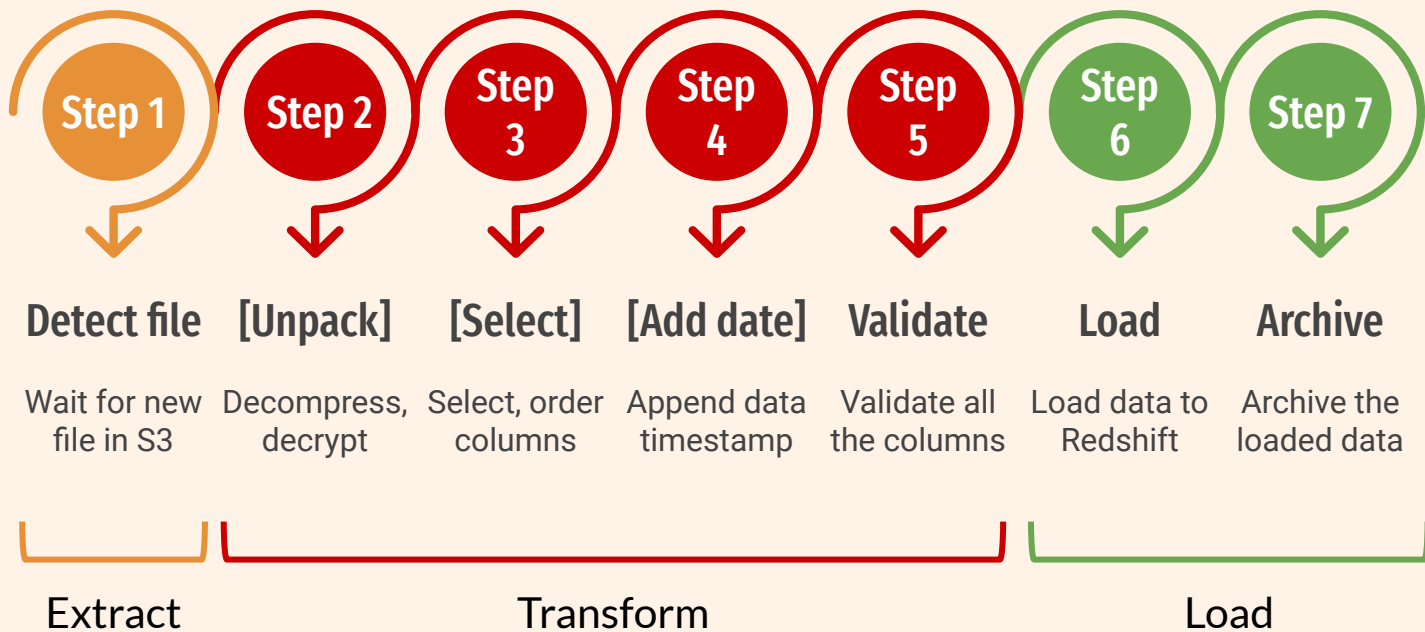
- Durable and flexible
- Many built-in functionalities
- Large community



FT Airflow setup

- Extension of the original Airflow project
- Developed and maintained by a dedicated FT Airflow team
- Additional built-in functionalities
- Provided to other FT teams as self-service

Extract, transform and load (ETL)



How we implemented them back then?

```
18     with FT DAG(dag_id='sfdc_case_cdc',
19                 schedule='0 23 * * *'):
20
21         detect_file = ETLFileIngestorSensorOperator(
22             task_id='detect_new_file',
23             timeout=10800,
24             poke_interval=900)
25
26         validation = ETLFileValidatorOperator(
27             task_id='validate_data',
28             data_input=StoragePath(task_id='transform_file'),
29             file_format=FileFormat.CSV,
30             file_delimiter=FileDelimiter.COMMA)
```

How we implemented them back then?

```
32     load_data_in_redshift = ETLS3ToRedshiftOperator(
33         task_id='load_data_to_redshift',
34         redshift_conn_id='redshift_conn_id',
35         table='ftsfv2db.sfdc_case_cdc',
36         s3_input=StoragePath(task_id='validate_data'),
37         copy_options=[
38             "DELIMITER ' '",
39             "EMPTYASNULL",
40             "CSV",
41             "timeformat 'auto'",
42             "TRUNCATECOLUMNS"
43         ])
44
45     archive_data_in_s3 = ETLS3ToS3Operator(
46         task_id='archive_file',
47         s3_input=StoragePath(task_id='validate_data'))
48
49     detect_file >> validation >> [load_data_in_redshift, archive_data_in_s3]
```

Challenges

- A lot of ETL pipelines to be migrated from the legacy tool
- Increasing requests for new ones
- Problem having different DAG configuration per environment
- **The pipelines look more or less the same**

We had to do something..

Dynamic ETL DAGs from configuration?



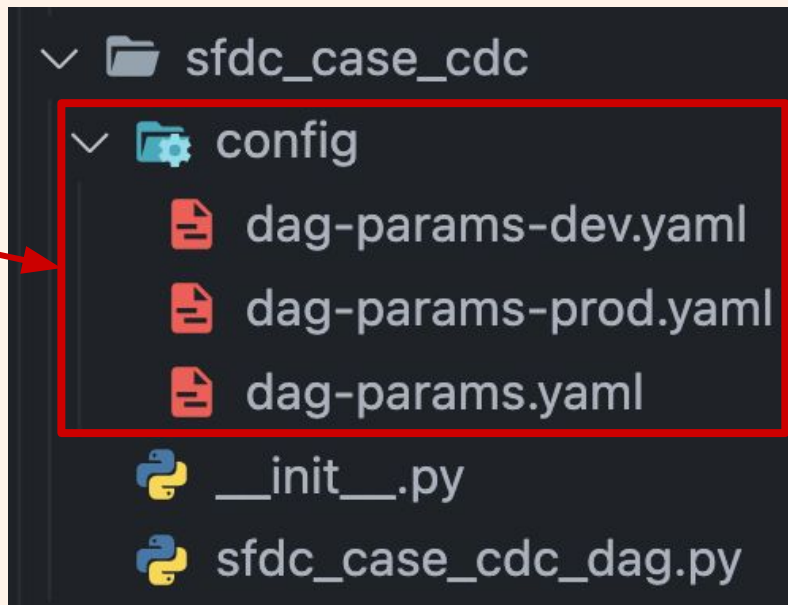
We decided to try it out..

Let's start with:

DAG Configuration system

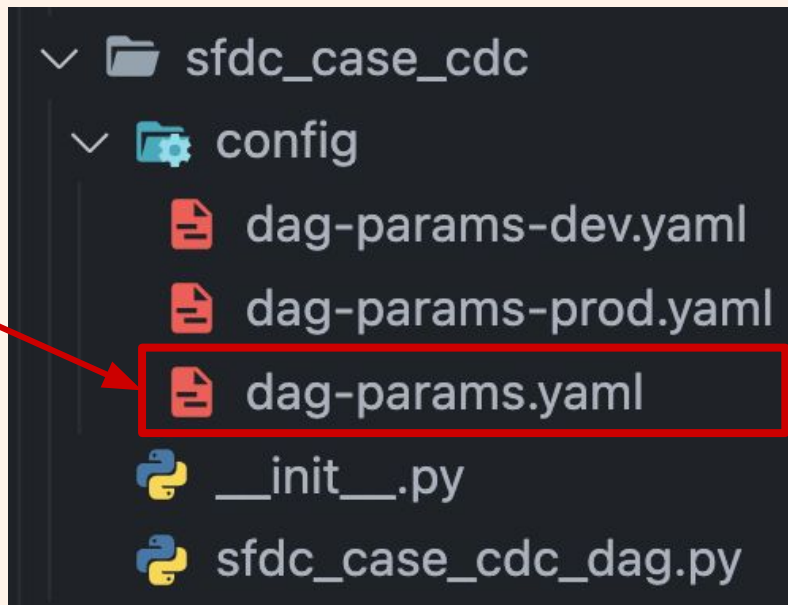
Separate DAG config per environment

DAG configuration
folder



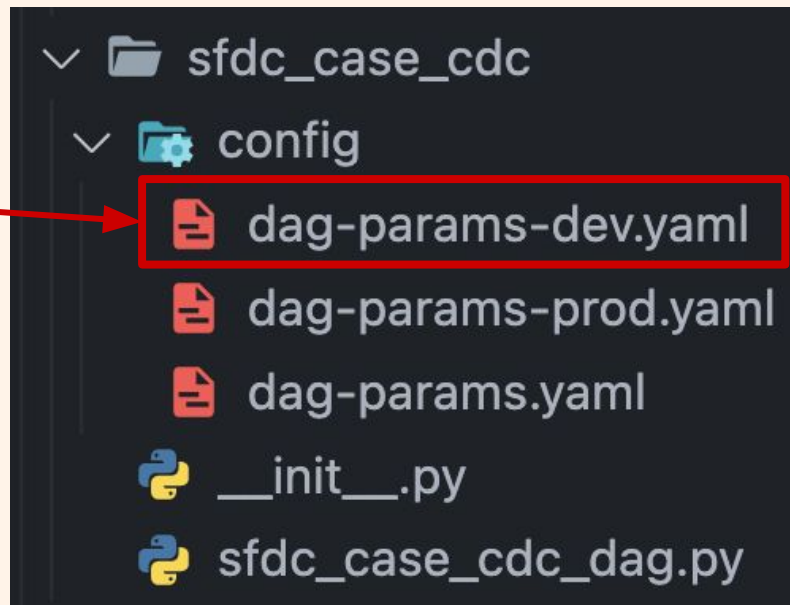
Separate DAG config per environment

Main DAG
configuration



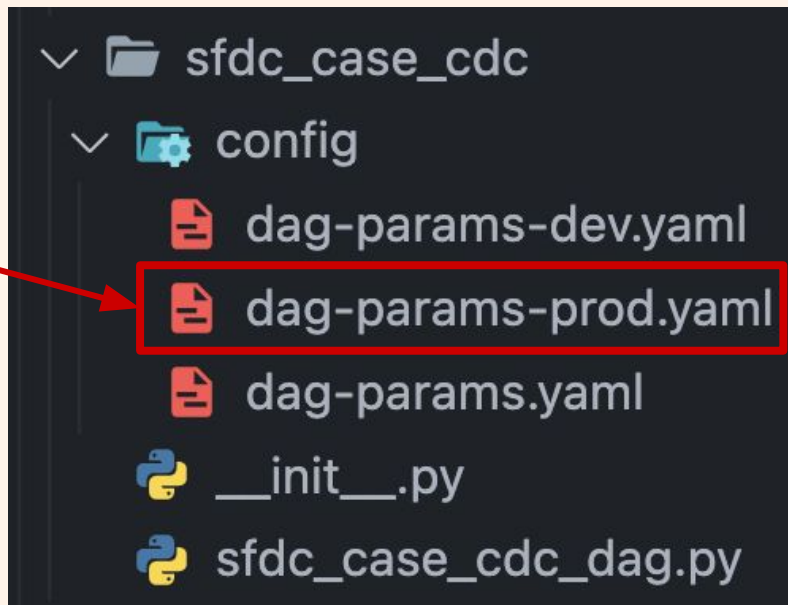
Separate DAG config per environment

DAG configuration
for DEV



Separate DAG config per environment

DAG configuration
for PROD



Example

Main config

```
detect_file:  
  poke_interval: 900  
  input:  
    bucket: ft.localdev  
    file_path: test/
```



DEV config

```
detect_file:  
  timeout: 10800  
  input:  
    bucket: ft.dev
```



Final config

```
detect_file:  
  timeout: 10800  
  poke_interval: 900  
  input:  
    bucket: ft.dev  
    file_path: test/
```

Example

Main config

```
detect_file:  
  poke_interval: 900  
  input:  
    bucket: ft.localdev  
    file_path: test/
```



DEV config

```
detect_file:  
  timeout: 10800  
  input:  
    bucket: ft.dev
```



Final config

```
detect_file:  
  timeout: 10800  
  poke_interval: 900  
  input:  
    bucket: ft.dev  
    file_path: test/
```

Usage inside DAG definition

```
with FTDAG(dag_id='sfdc_case_cdc',
           max_active_runs=1,
           schedule='0 23 * * *') as dag:

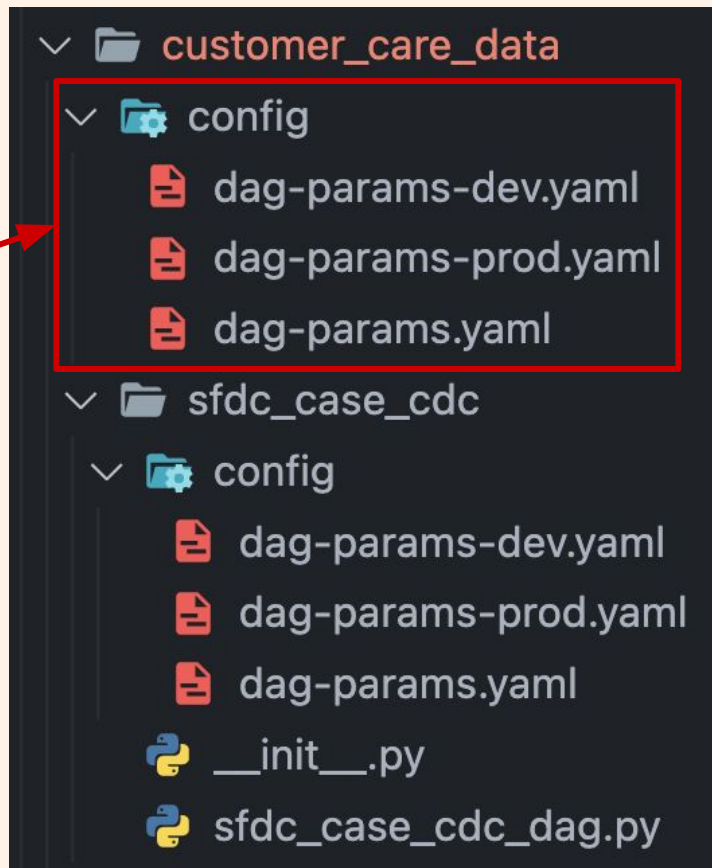
    detect_file = ETLFileIngestorSensorOperator(
        task_id=dag.dag_params.tasks.detect_file.task_id,
        timeout=dag.dag_params.tasks.detect_file.timeout,
        poke_interval=dag.dag_params.tasks.detect_file.poke_interval,
        s3_input=StoragePath(connection_id=dag.team_params.connections,
                              bucket=dag.dag_params.tasks.detect_file.bucket,
                              file_path=dag.dag_params.tasks.detect_file.file_path)
```

One step further:

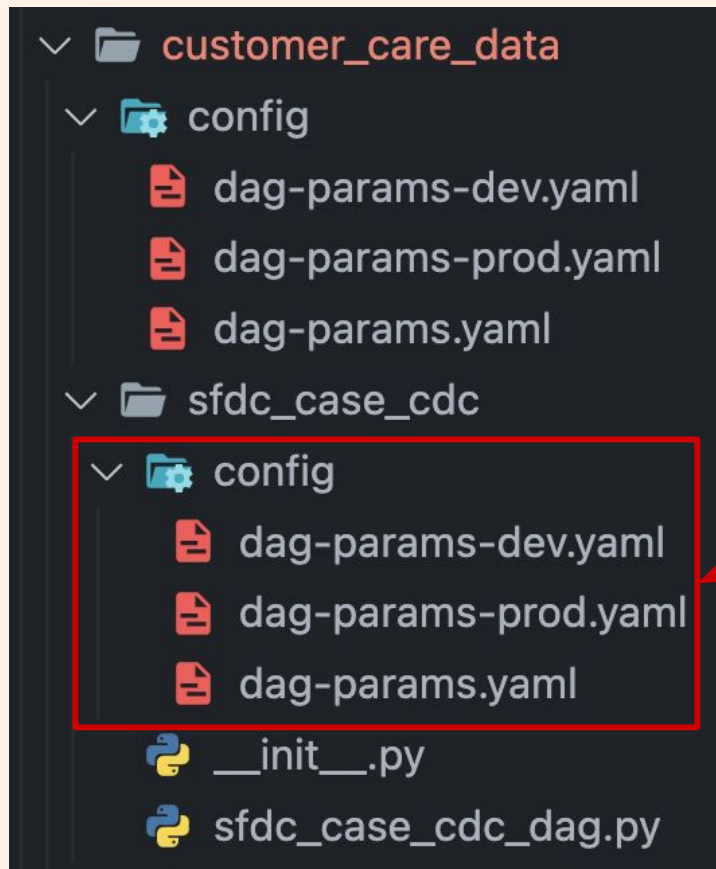
Nested DAG Configurations

Nested DAG configs

Parent folder
configuration

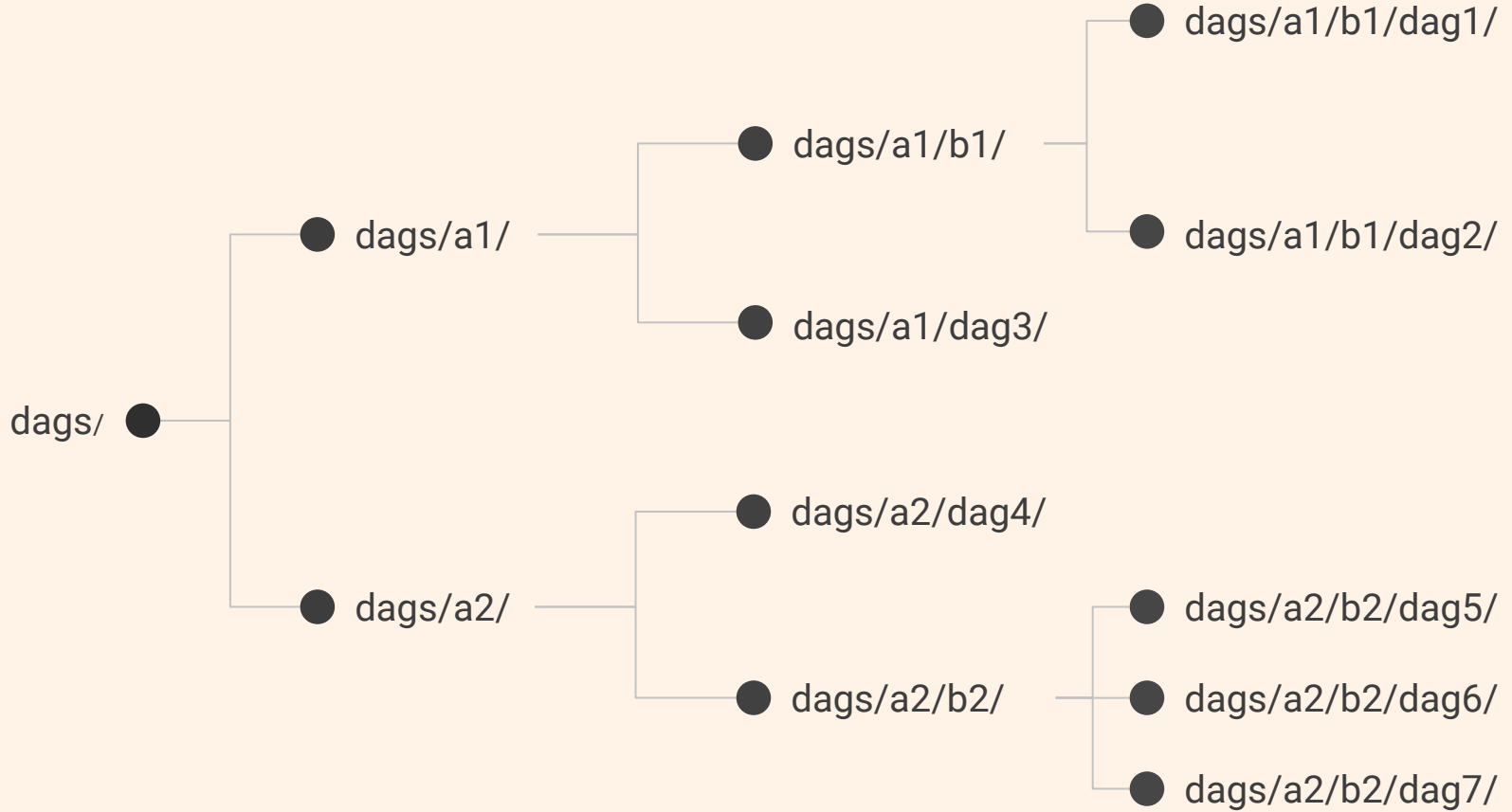


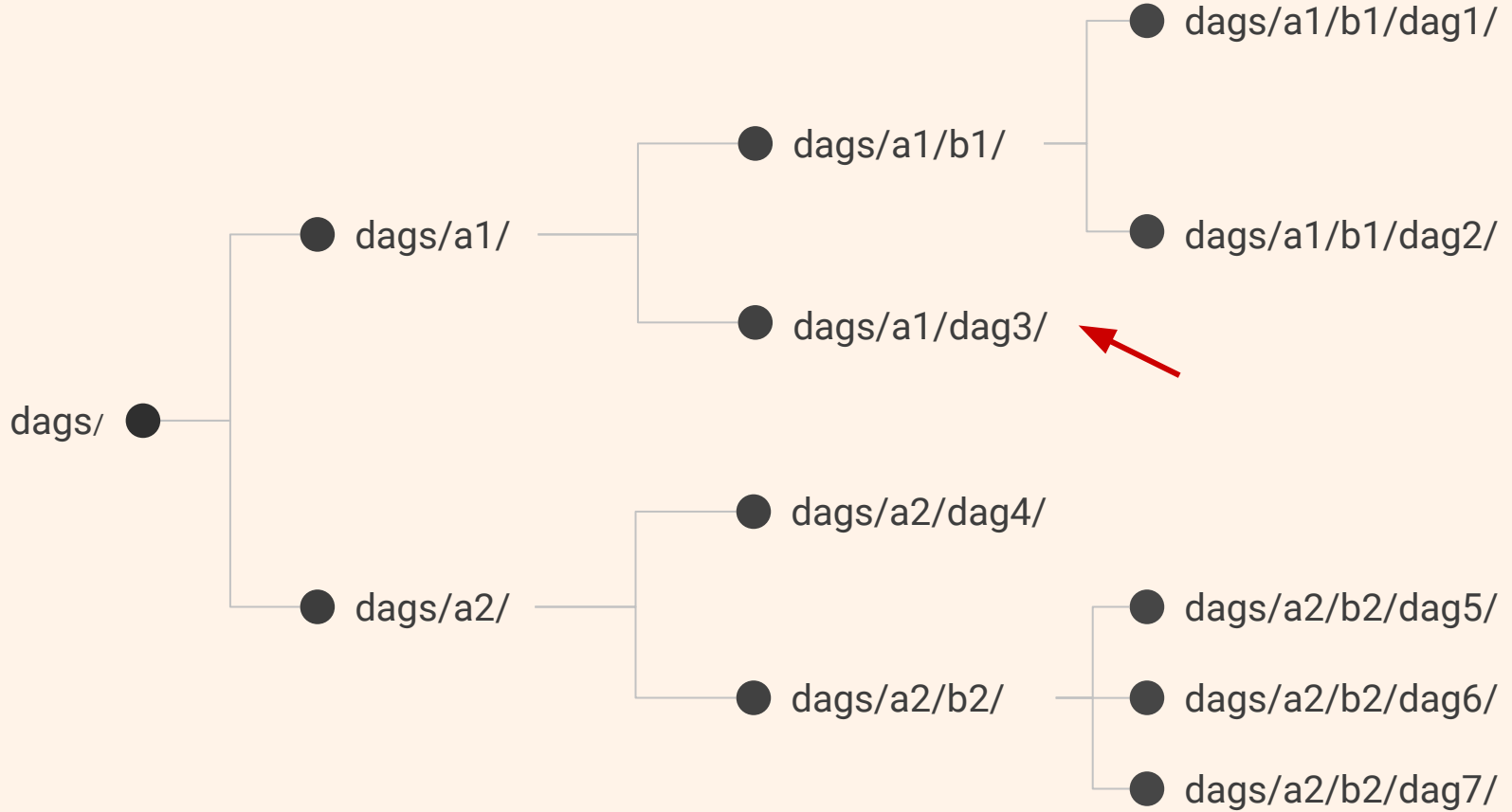
Nested DAG configs

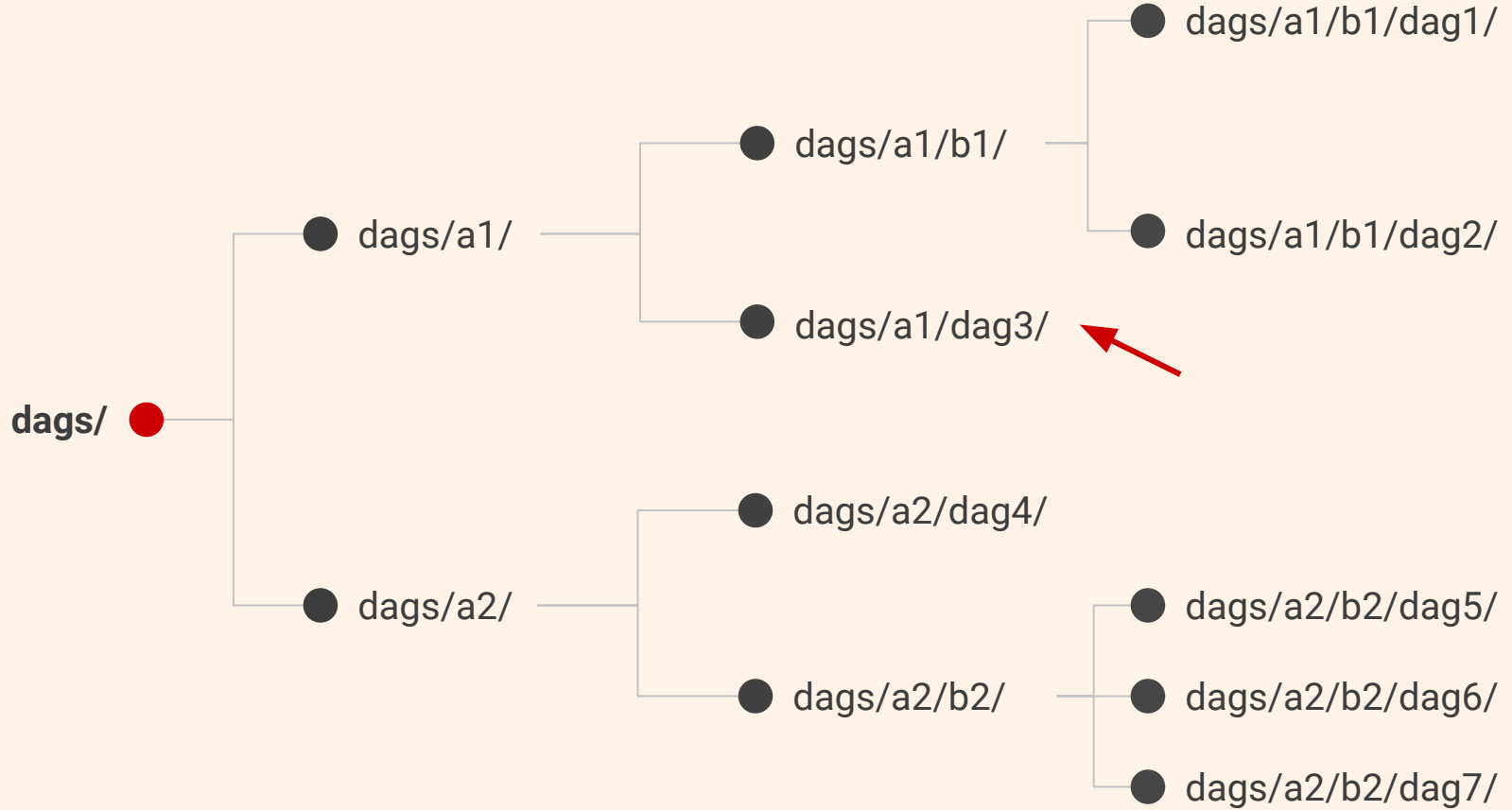


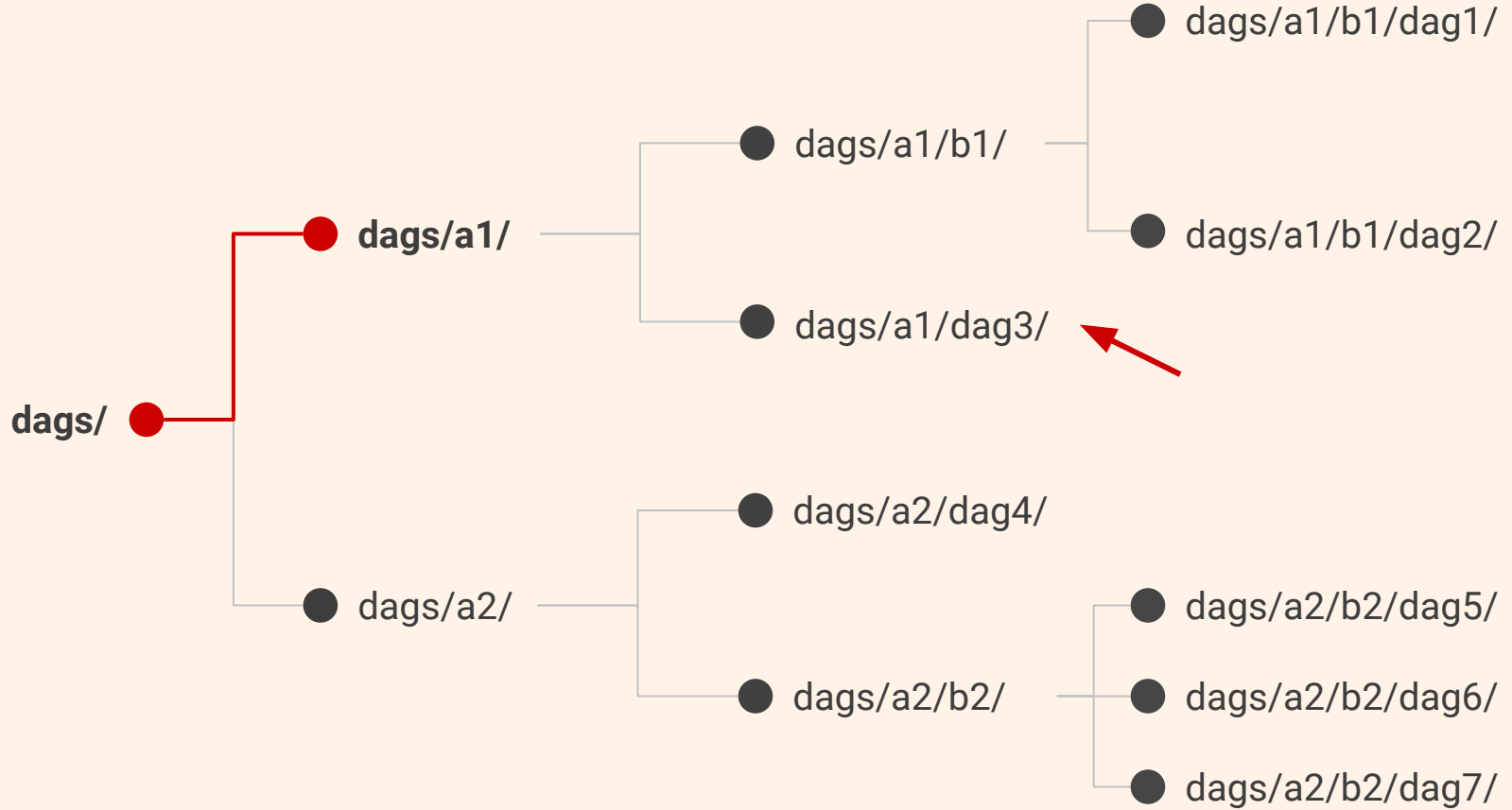
DAG folder
configuration

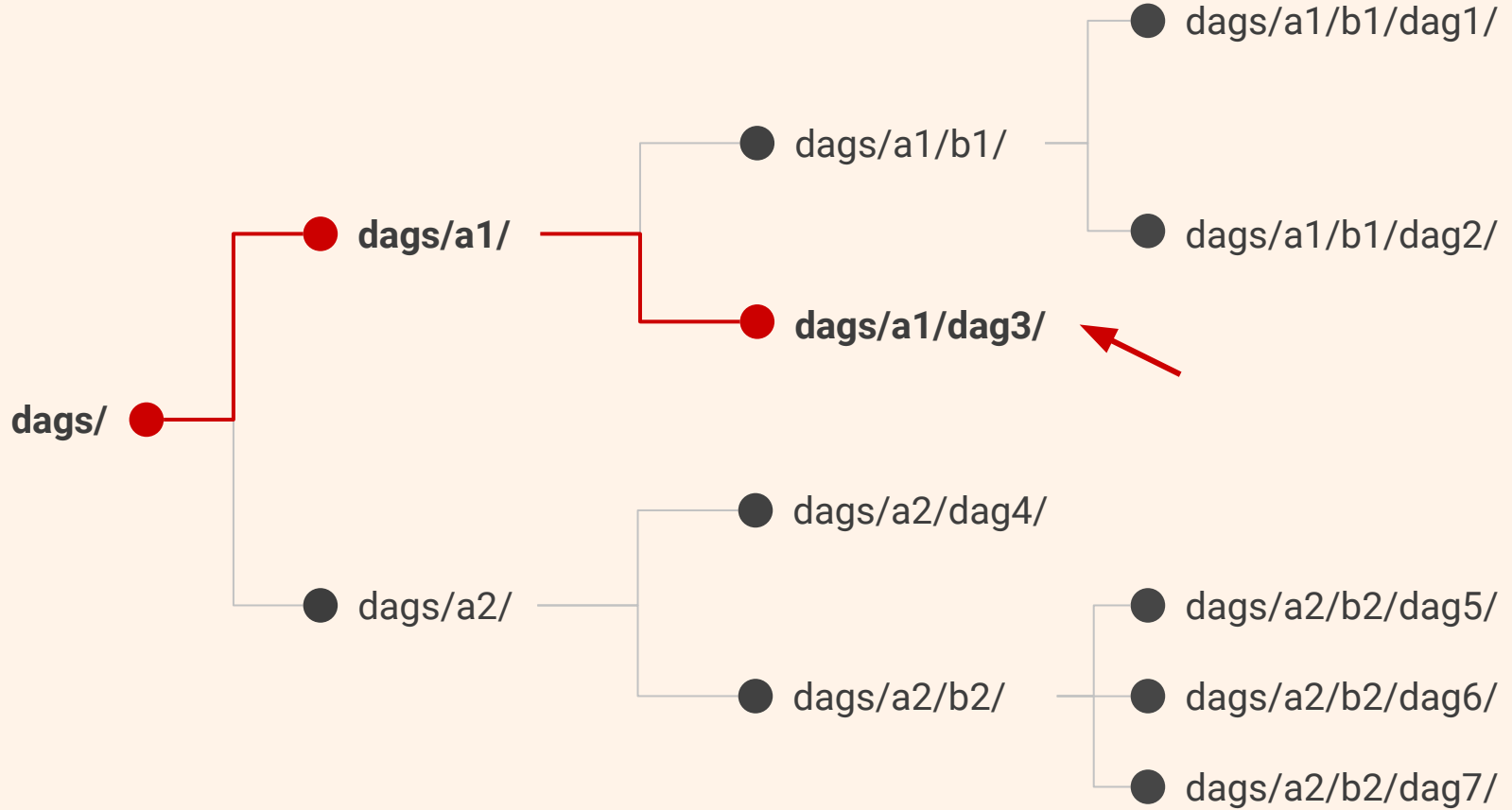
Let's see some examples..

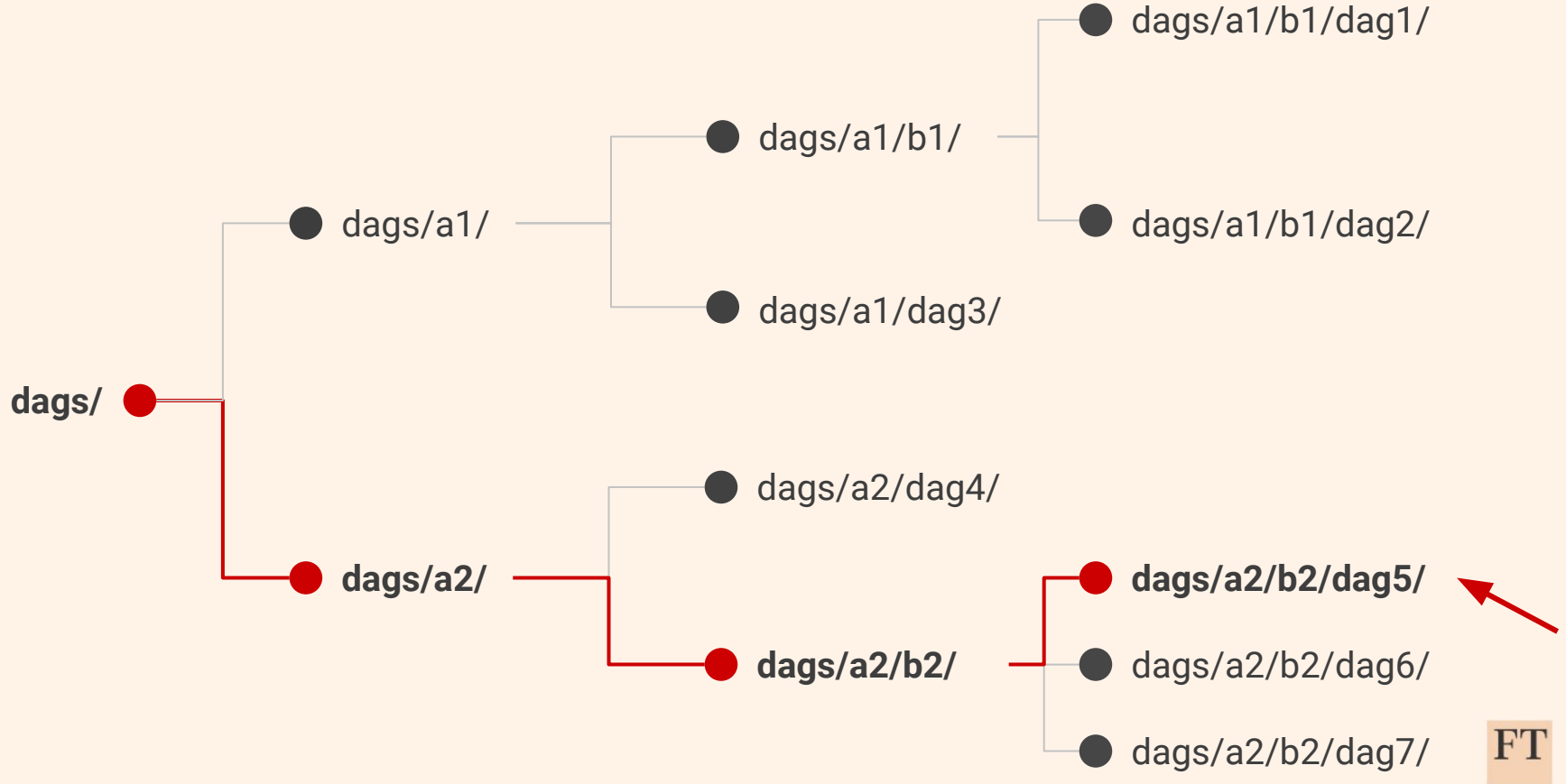












What's the final config?!



DAG: dataplatform_sfcd_case_cdc

Local

```
etl:
  archive:
    s3_output:
      bucket: localdev.ft.dw.archive
      connection_id: aws_conn_id
      sub_path: '{{dag.dag_id}}/{{logical_date}}'
  tasks:
    archive_data_in_s3:
      output:
        bucket: localdev.ft.dw.archive
        file_path: dataplatform/sfdc_archive/sfdc_case_cdc
      task_id: archive_files_for_storage
    call_stored_procedure:
      parameters:
        business_key: id
        job_type: IU
        row_order: id
      task_id: staging_to_cdc
  common:
    job_id: sfdc_case_cdc
    redshift:
      cdc_table: sfdc_case_cdc
      schema: ftsfv2db
      staging_table: sfdc_case_stg
    detect_file:
      input:
        bucket: ip-crm-data-bridge-dev
        file_path: Exports/dw/Global-case/
      mode: reschedule
      poke_interval: 900
      soft_fail: true
```

Dev

```
etl:
  archive:
    s3_output:
      bucket: dev.ft.dw.archive
      connection_id: aws_conn_id
      sub_path: '{{dag.dag_id}}/{{logical_date}}'
  tasks:
    archive_data_in_s3:
      output:
        bucket: dev.ft.dw.archive
        file_path: dataplatform/sfdc_archive/sfdc_case_cdc
      task_id: archive_files_for_storage
    call_stored_procedure:
      parameters:
        business_key: id
        job_type: IU
        row_order: id
      task_id: staging_to_cdc
  common:
    job_id: sfdc_case_cdc
    redshift:
      cdc_table: sfdc_case_cdc
      schema: ftsfv2db
      staging_table: sfdc_case_stg
    detect_file:
      input:
        bucket: ip-crm-data-bridge-dev
        file_path: Exports/dw/Global-case/
      mode: reschedule
      poke_interval: 900
      soft_fail: true
```

Prod

```
etl:
  archive:
    s3_output:
      bucket: prod.ft.dw.archive
      connection_id: aws_conn_id
      sub_path: '{{dag.dag_id}}/{{logical_date}}'
  tasks:
    archive_data_in_s3:
      output:
        bucket: prod.ft.dw.archive
        file_path: dataplatform/sfdc_archive/sfdc_case_cdc
      task_id: archive_files_for_storage
    call_stored_procedure:
      parameters:
        business_key: id
        job_type: IU
        row_order: id
      task_id: staging_to_cdc
  common:
    job_id: sfdc_case_cdc
    redshift:
      cdc_table: sfdc_case_cdc
      schema: ftsfv2db
      staging_table: sfdc_case_stg
    detect_file:
      input:
        bucket: ip-crm-data-bridge-dev
        file_path: Exports/dw/Global-case/
      mode: reschedule
      poke_interval: 900
      soft_fail: true
```

And what's next?

Defining a common structure:

ETL Pipeline Config

Structure

```
job_id: salesforce_contact_v2_cdc  
vendor: 'salesforce_v2'
```

```
meta:
```

```
  header_lines_count: 1  
  threshold_percentage: 0  
  file_delimiter: ','
```

```
file_ingest:
```

```
  poll_interval: 900 # 15 minutes  
  timeout: 10800 # 3 hours
```

```
  entrypoint_path:
```

```
    connection_id: 'aws_conn_id_crm_role_prod'  
    bucket: 'ip-crm-data-bridge-prod'  
    file_path: dw/Global-contact/
```

Structure

```
columns:
  - name: id
    data_type: varchar
    max_length: 18
    nullable: true
  - name: isdeleted
    data_type: boolean
    nullable: true
  - name: lastmodifieddate
    data_type: timestamp
    data_format: '%Y-%m-%dT%H:%M:%S.%f%z'
    nullable: true
  - name: annualrevenue
    data_type: numeric
    precision: 18
    scale: 2
    should_round: true
    nullable: true
```

Structure

```
redshift:
  schema: ftsalesforcev2db
  final_table: salesforce_contacts_v2_cdc
  copy_options:
    - EMPTYASNULL
    - timeformat 'auto'
    - TRUNCATECOLUMNS
    - dateformat 'auto'

archive:
  s3_output:
    connection_id: aws_conn_id
    bucket: prod.ft.dw.archive
    file_path: dataplatform/salesforce_v2_archive
    sub_path: '{{dag.dag_id}}/{{logical_date}}'
  s3_output_encryption_config:
    keys_to_use: ftwebanalytics
```


Time for automation..

ETL Reusable DAG

What is the idea behind it?

1. Read the ETL Pipeline config

What is the idea behind it?

1. Read the ETL Pipeline config
2. Create the DAG

Implementation

```
@classmethod
def build_dag(cls,
              dag_configuration: Dict,
              default_args: Dict) -> FTDAG:
    default_args = default_args or {}
    default_args['depends_on_past'] = default_args.get('depends_on_past', True)
    default_args['wait_for_downstream'] = default_args.get('wait_for_downstream', True)
    default_args = DAGArgumentsBuilder.build(**default_args)

    with FTDAG(**dag_configuration,
               default_args=default_args) as dag:
        cls._build_tasks(dag=dag)
    return dag
```

What is the idea behind it?

1. Read the ETL Pipeline config
2. Create the DAG
3. Add the necessary operators

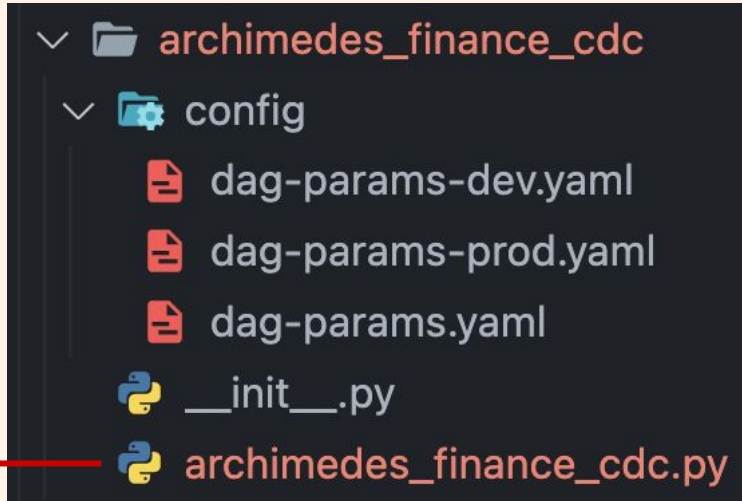
Implementation

```
@classmethod
def _build_tasks(cls,
                 dag: FTDAG,
                 entrypoint_path: Optional[StoragePath] = None,
                 is_task_group: bool = False) -> None:
    job_config = ETLConfigService.read_job_config_from_dag_params(dag_params=dag.dag_params)
    tasks = []

    if not is_task_group:
        task = cls._entry_task_factory(job_config)
        tasks.append(task)

    if job_config.get_transformations():
        file_transformer = ETLFileTransformerOperator(
            task_id='transform',
            data_input=StoragePath(task_id=tasks[-1].task_id) if tasks else entrypoint_path
        )
        tasks.append(file_transformer)
```

Usage with predefined data sources



```
ETLReusableDAG.build_dag(  
    dag_configuration={'dag_id': 'archimedes_finance_cdc',  
                        'schedule': '0 10 * * *',  
                        'tags': ['dataplatfrom', 'ingester', 'archimedes', 'finance']},  
    default_args={'start_date': datetime(2023, 7, 28, 11, 0, 0)}  
)
```


Usage with custom data sources

```
with FTDAG(dag_id="appsflyer_aggregated_skan_installs",
           schedule='0 1 * * *',
           start_date=datetime.datetime(2022, 8, 30, 0, 0, 0)) as dag:

    http_to_s3 = HttpToS3Operator(task_id=dag.dag_params.http_to_s3.task_id,
                                  http_conn_id=dag.dag_params.http_to_s3.http_conn_id,
                                  endpoint=dag.dag_params.endpoint)

    TG = ETLReusableDAG.build_task_group(
        dag=dag,
        entrypoint_path=StoragePath(task_id=dag.dag_params.http_to_s3.task_id)
    )

    http_to_s3 >> TG
```

And even more automation..

Generation of DDL SQLs

YAML

```
redshift:
  final_table: finance_cdc
  stg_table: finance_stg
  business_key: financeid
```

columns:

- name: addressid
data_type: bigint
nullable: true
- name: product
data_type: varchar
max_length: 110
nullable: true
- name: subcounter
data_type: bigint
nullable: true



SQL

```
CREATE TABLE ftarchimedesdb.finance_stg
(
  addressid BIGINT
  ,product VARCHAR(110)
  ,subcounter BIGINT
  ,"type" VARCHAR(110)
  ,financeid BIGINT
  ,createdate DATE
  ,periodfrom DATE
```

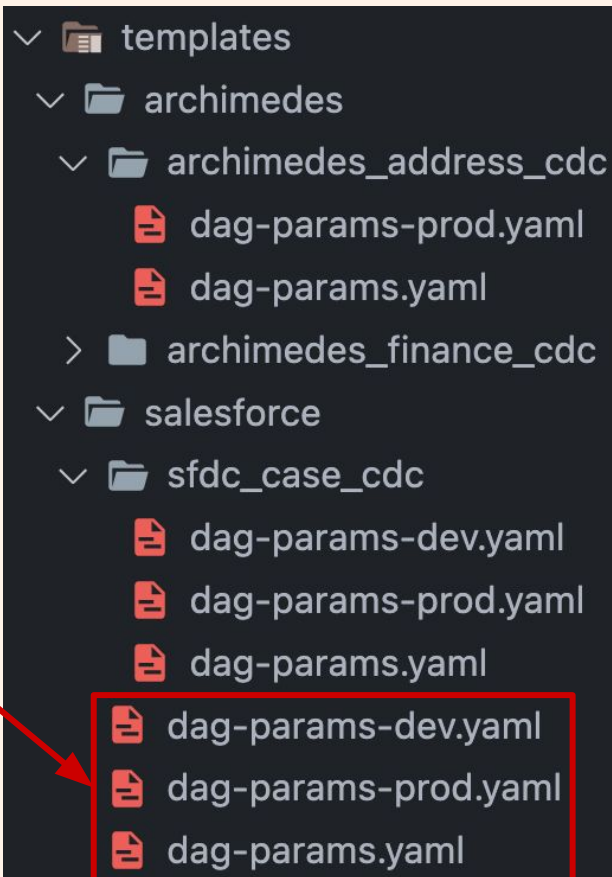
Let's recap..

- Flexible DAG config system
- Additional ETL feature
 - Automates the DAG creation
 - Enforces the same config structure throughout all the DAGs
 - Removes most of the repetitive configuration and DAG code
 - Easily apply changes to all DAGs (can be a problem as well)
 - Opens the door for more automations

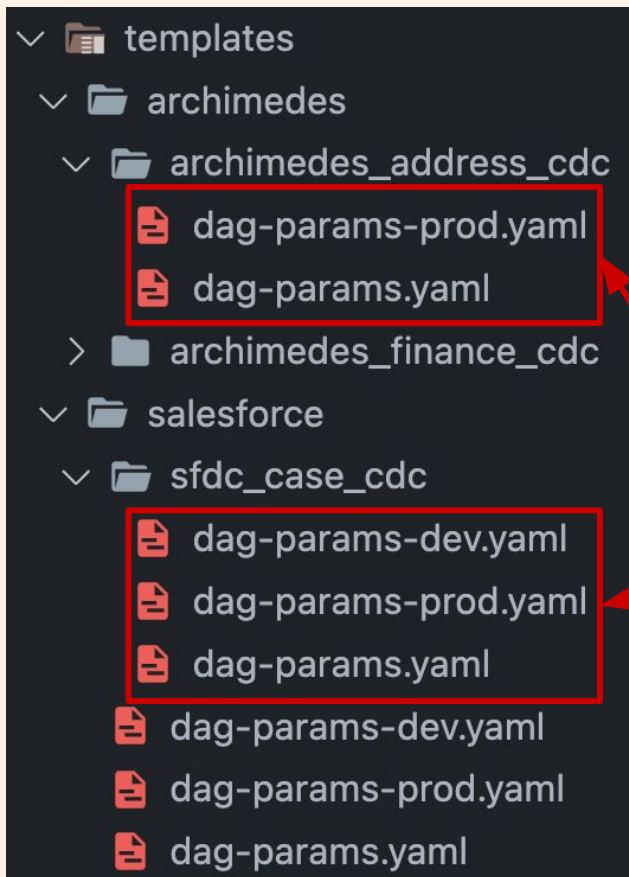
Further improvements

Step 1: No DAG code at all

Parent folder
configuration

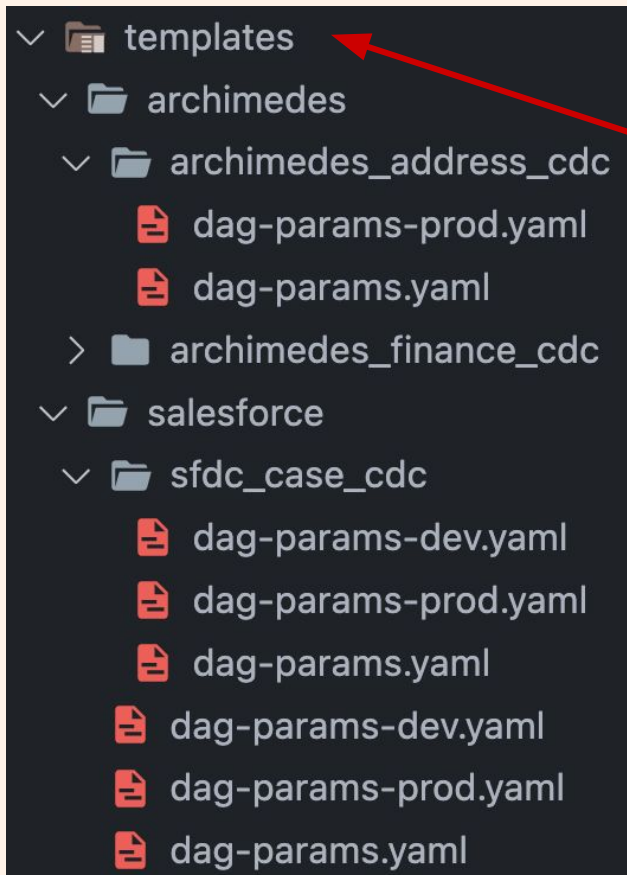


Step 1: No DAG code at all



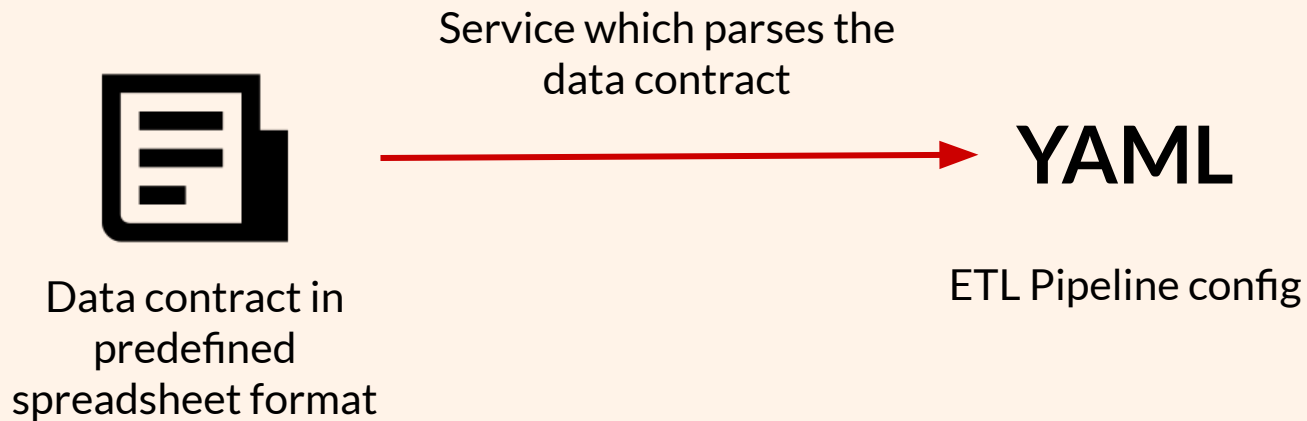
DAG folder
configuration

Step 1: No DAG code at all

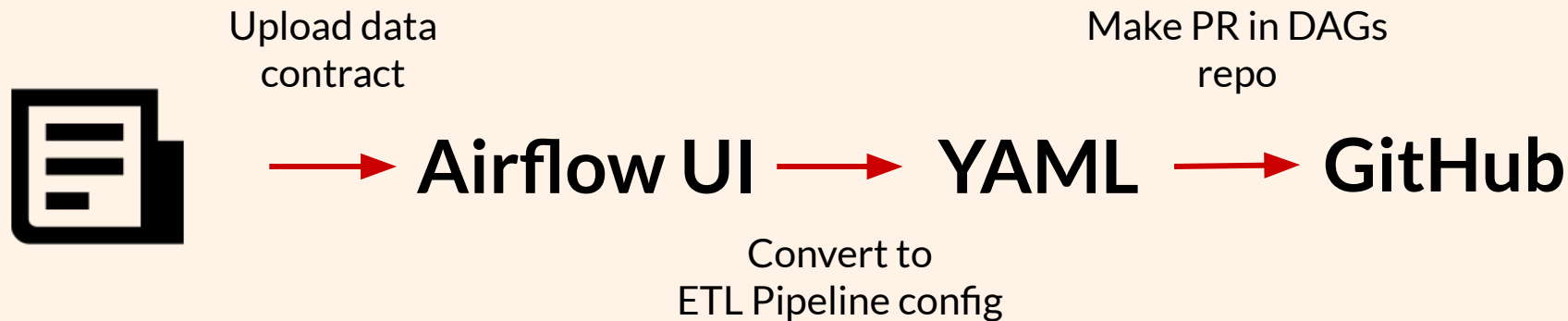


Service which scans
the folder and
creates the DAGs

Step 2: Convert data contract to ETL Pipeline config



Step 3: Connect everything and make a GitHub PR





Yes!

ETL DAGs can be written just from configuration!

Questions?

Email: data.platform.airflow@ft.com

LinkedIn: [Zdravko Hvarlingov](#) & [Vladi Nekolov](#)