Airflow in an On-premise Data Mesh Setup

Speaker

Jorrick Sleijster
Data Engineer
Data Platform Development
Tennis & Padel tennis

Adyen

We provide a single payment platform globally to accept payments and grow revenue online, on mobile, and at the point of sale

FACEBOOK

Bla Bla Car

AliExpress™

LinkedIn

Uber

Dropbox

Evernote

Microsoft

Spotify

ebay

The Workflow of the Big Data Platform

Fully on-premise
We have all servers in-house. We manage everything from hardware to software. Different clusters per env: beta, test and live.

Weekly releases
Each week we start with a new release on beta. Two days later deployed to test, and finally a week later to live.

Teams / Streams
We have 20+ streams and 100+ data scientists on the big data platform. Streams are responsible for their own DAGs.

Airflow Cluster Setup
at Adyen

Airflow 2.2

Celery workers

Yarn Queue

Postgres DB

adyen extensions

Models, views, database listeners

adyen

Apache Airflow

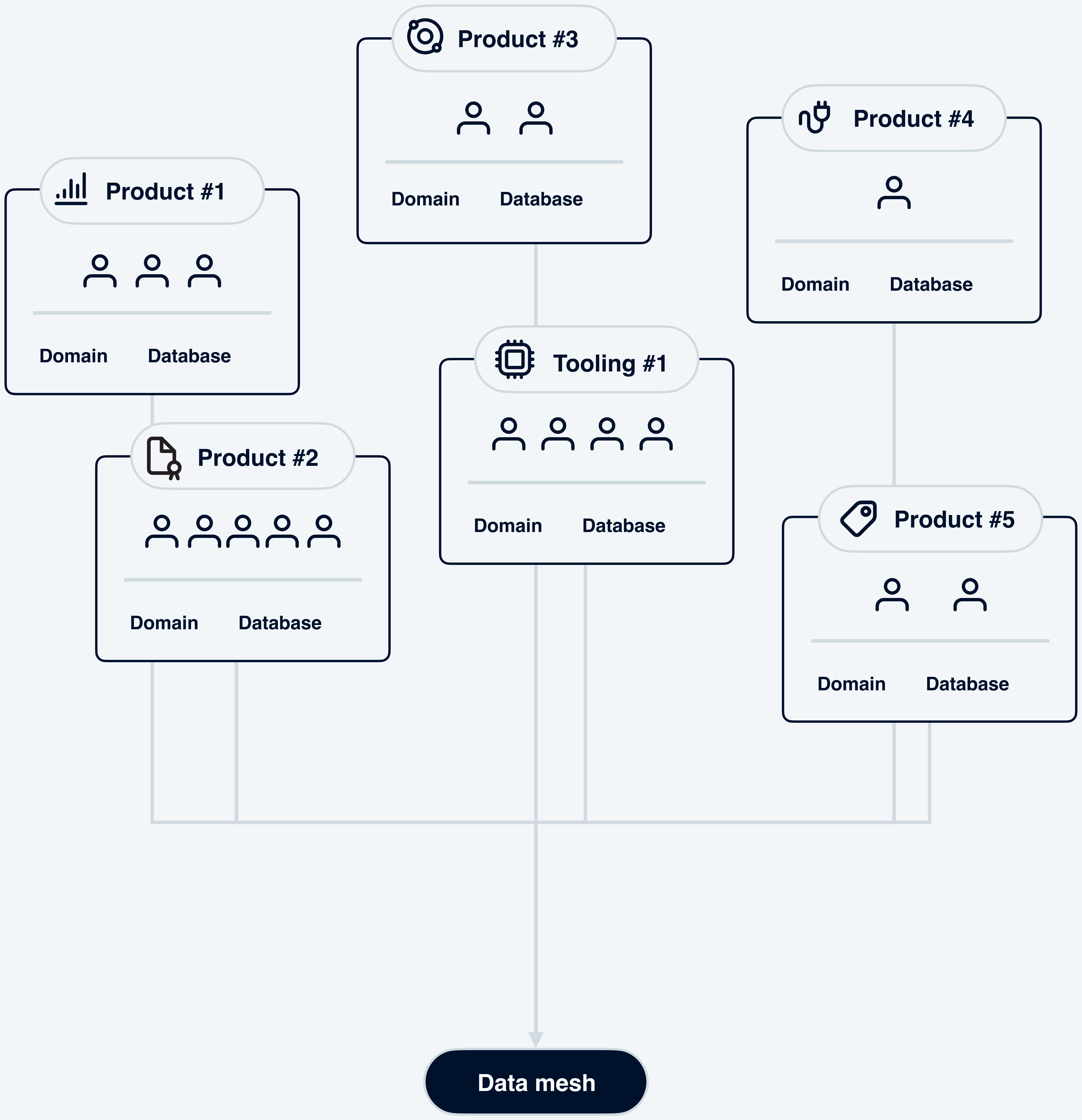
Why Data Mesh

- What is a Data Mesh?
  - Monolithic data infrastructure
  - Distributed domains
  - Domain responsible for their own ETLs & data

- Product teams(domains) with a clear focus
  - Front-enders, back-enders, data scientists

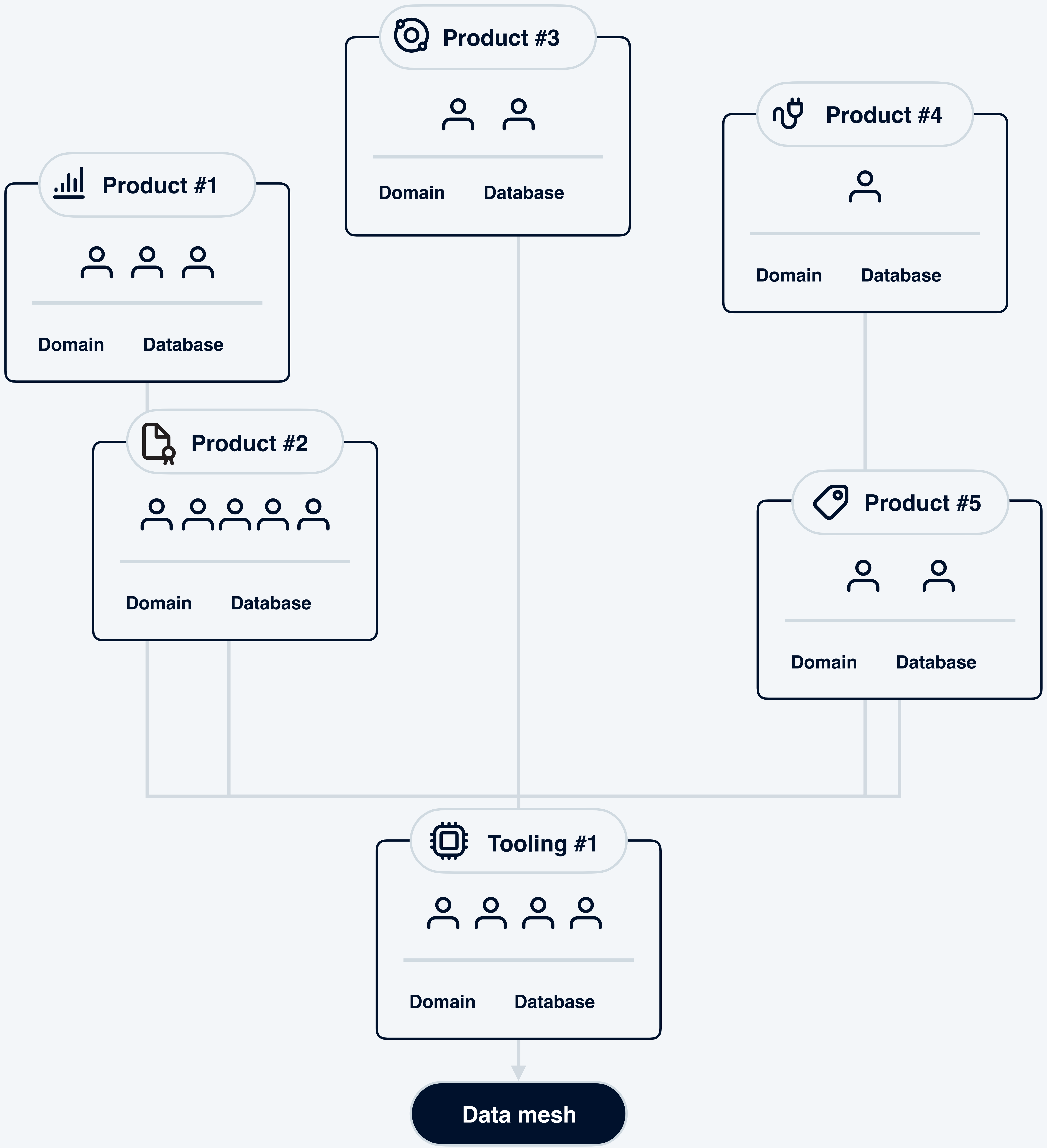
- No central ETL team

- Self service



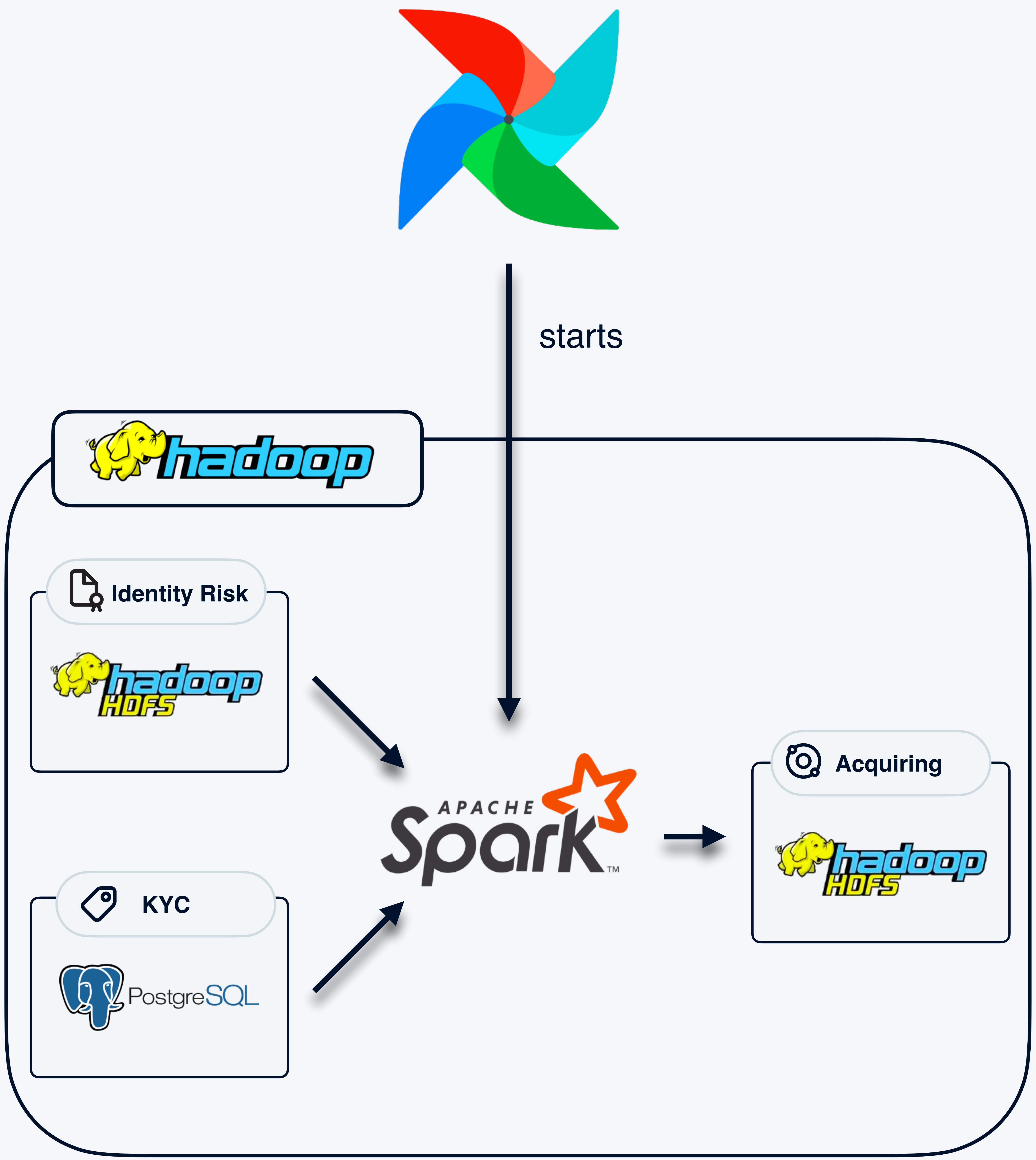
Data Mesh Teams

- Product teams
- Tooling teams
- Infrastructure teams



Creating a (new) Data Pipeline

- Create an Airflow DAG
- Create the ETL code



Creating a (new) Data Pipeline

- Create an Airflow DAG
- Create the ETL code

- We need to:
  - Define the schema
  - Have enough resources for the ETL
  - Possibly handle updates of the schema
  - Retention, we don't have infinite storage
  - Remove the table
  - Don't want to (accidentally?) modify other teams resources.

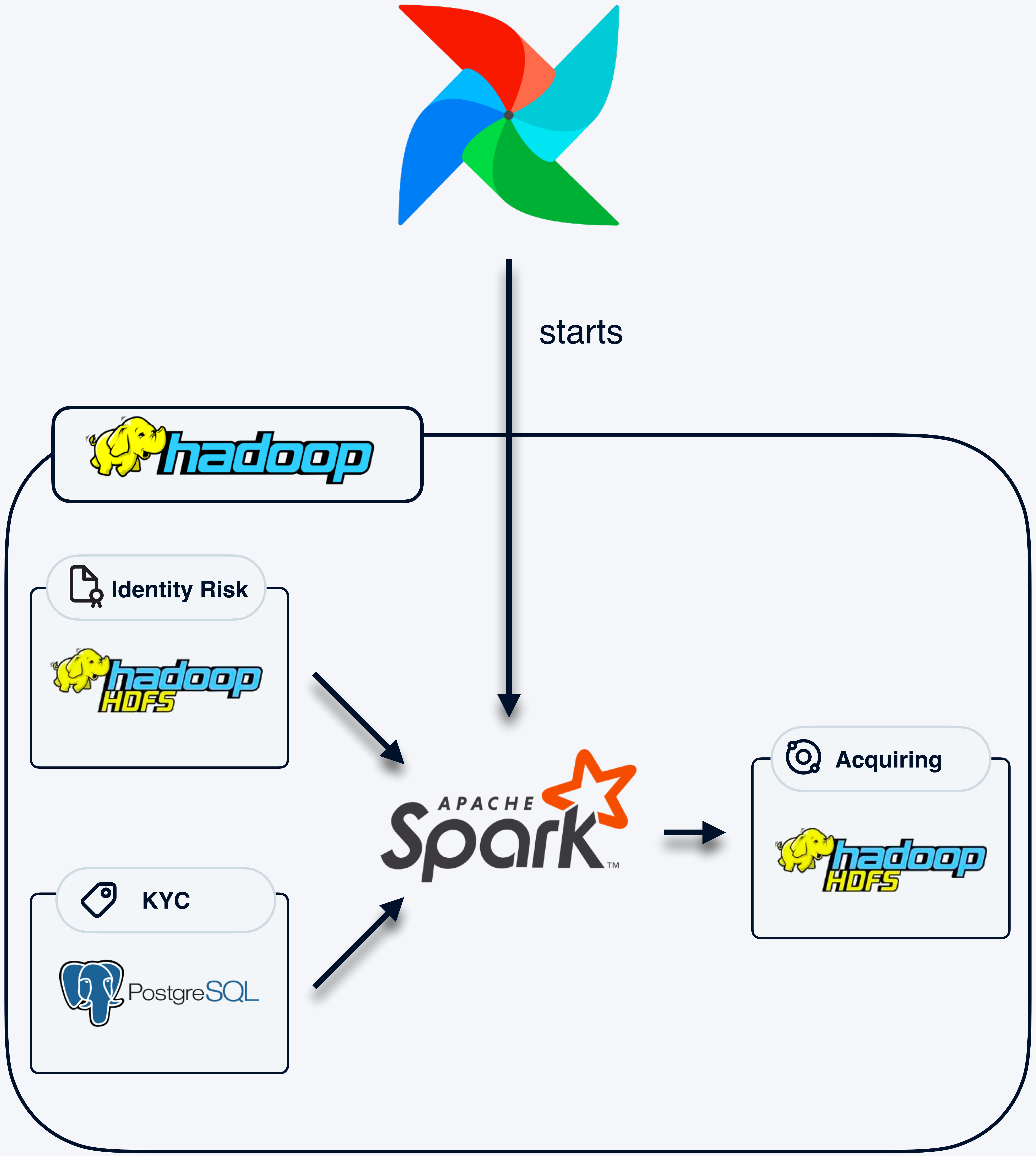


Table Schemas

How can we enable schema evolution?

Tuning Resources

How can we handle data outgrowing the predefined ETL resources?

Retention Period

How can we save HDFS from being filled with data we don't need anymore?

User Permissions

How can we give users access to only the parts that they need?

Table Schemas

- We are managing an ETL pipeline
  - How & where do we define the schema?
  - How do we update the schema?
  - How do we delete tables we no longer need?

```sql
CREATE TABLE `parquetfilesize` (
  `file_size` BIGINT,
  `database` STRING,
  `table_name` STRING,
  `file_path` STRING,
  `human_readable_size` STRING,
  `write_timestamp` BIGINT,
  `partition_date` STRING,
  `snapshot_timestamp` STRING,
)
USING parquet
OPTIONS (
  `serialization.format` '1',
  path '{path}'
)
PARTITIONED BY (partition_date)
TBLPROPERTIES (
  sort_keys = 'partition_date,database,table_name',
  repartition_factor = 1
)
```

Table Schemas Library

- Each domain has their own database.

- Each database has multiple scopes:
  - pii - Requires white-listed access.
  - private - Access by domain members only.
  - public - Access by everyone.

- Each database scope can have multiple tables.

- Each table has at least one change file.

- Change types: new, (in)compatible, remove

Tool to create schemas from a Spark DataFrame.

```
lib_table_schemas
  analytics_core
    public
      changes
        V1_10-NEW-sparkpushedfilters-homer.ddl
        V1_12-INCOMPATIBLE-sparkpushedfilters-marge.ddl
        V1_12-NEW-tablemetadata-marge.ddl
      sparkpushedfilters.ddl
      tablemetadata.ddl
  cdi
    pii
      changes
        V1_10-NEW-pii_table-marge.ddl
        V1_12-INCOMPATIBLE-pii_table-lisa.ddl
      pii_table.ddl
    public
      changes
        V1_11-NEW-table_compaction_experiment-jorrick.ddl
      table_compaction_experiment.ddl
  monitoring
    public
      changes
        V1_7-NEW-apicall-homer.ddl
        V1_9-NEW-api_performance-bart.ddl
        V1_10-INCOMPATIBLE-api_performance-bart.ddl
        V1_12-REMOVE-api_performance-bart.ddl
        V1_13-COMPATIBLE-apicall-homer.ddl
      apicall.ddl
```

Process Flow

1. Update the schema file (.ddl)
2. Introduce the schema change file (.ddl)
3. Create an MR (merge request)
4. Teammate reviews and approves MR
5. MR is merged

```
CREATE TABLE `parquetfilesize` (
  `file_size` BIGINT,
  `database` STRING,
  `table_name` STRING,
  `file_path` STRING,
  `human_readable_size` STRING,
  `write_timestamp` BIGINT,
  `partition_date` STRING,
  `snapshot_timestamp` STRING,
)
USING parquet
OPTIONS (
  `serialization.format` '1',
  path '{path}'
)
PARTITIONED BY (partition_date)
TBLPROPERTIES (
  sort_keys = 'partition_date,database,table_name',
  repartition_factor = 1
)
```

Process Flow

1. Update the schema file (.ddl)
2. Introduce the schema change file (.ddl)
3. Create an MR (merge request)
4. Teammate reviews and approves MR
5. MR is merged
6. Release a new version on the cluster
7. Duty rolls out all schema change files for release
8. (Optional) User can also perform ad-hoc table changes like redefining the table with the latest schema.

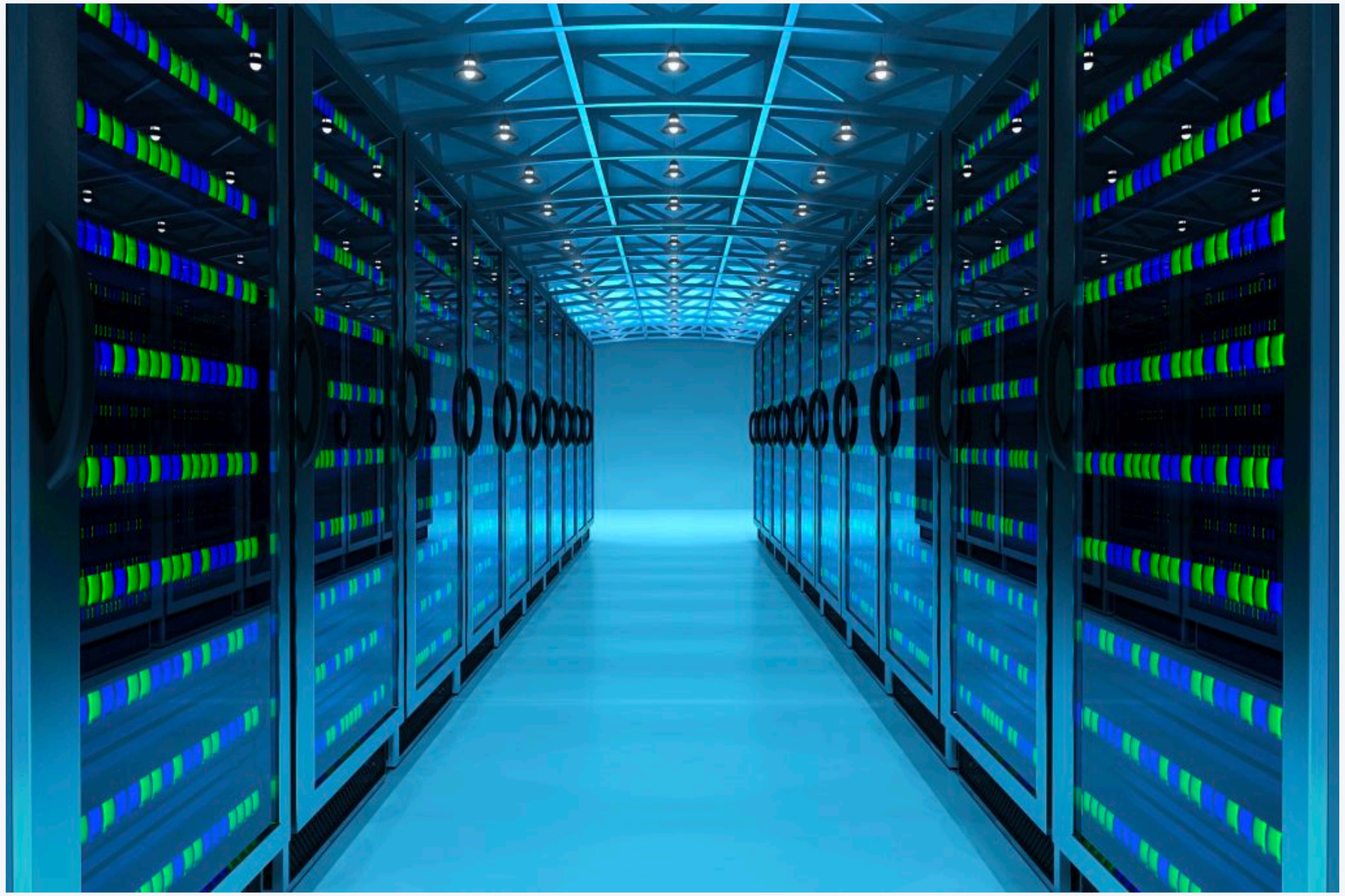
not-so-live demo

adyen


localhost
Airflow
Airflow
DAGs
Security
Browse
Admin
Docs
Duty
Cluster: dev-local
10:30 UTC
AA
Table management View
Logs
roll_out_new_table_changes_and_avro_tables 2022-05-13T05:12:59
Ad-hoc Table Changes
Hive Database
Table Name
Table Privacy: Public Private PII
Delete hive table
Delete table data
Define hive table & Do MSCK repair
Roll Out Change
Roll out Table Changes
Please select the changes which you want to roll out from the table below.
Filter on release version
Select a release
Filter on other properties
Filter changes
Reset selection
Roll Out Changes
Showing 1 to 10 of 10 entries
RC Change Type Database Table Privacy Table Name Requested by Status
V1_12 REMOVE monitoring public api_performance bart -
V1_12 NEW analytics_core public tablemetadata marge -
V1_12 INCOMPATIBLE cdi pii pii_table lisa -
V1_12 COMPATIBLE analytics_core public sparkpushedfilters marge -

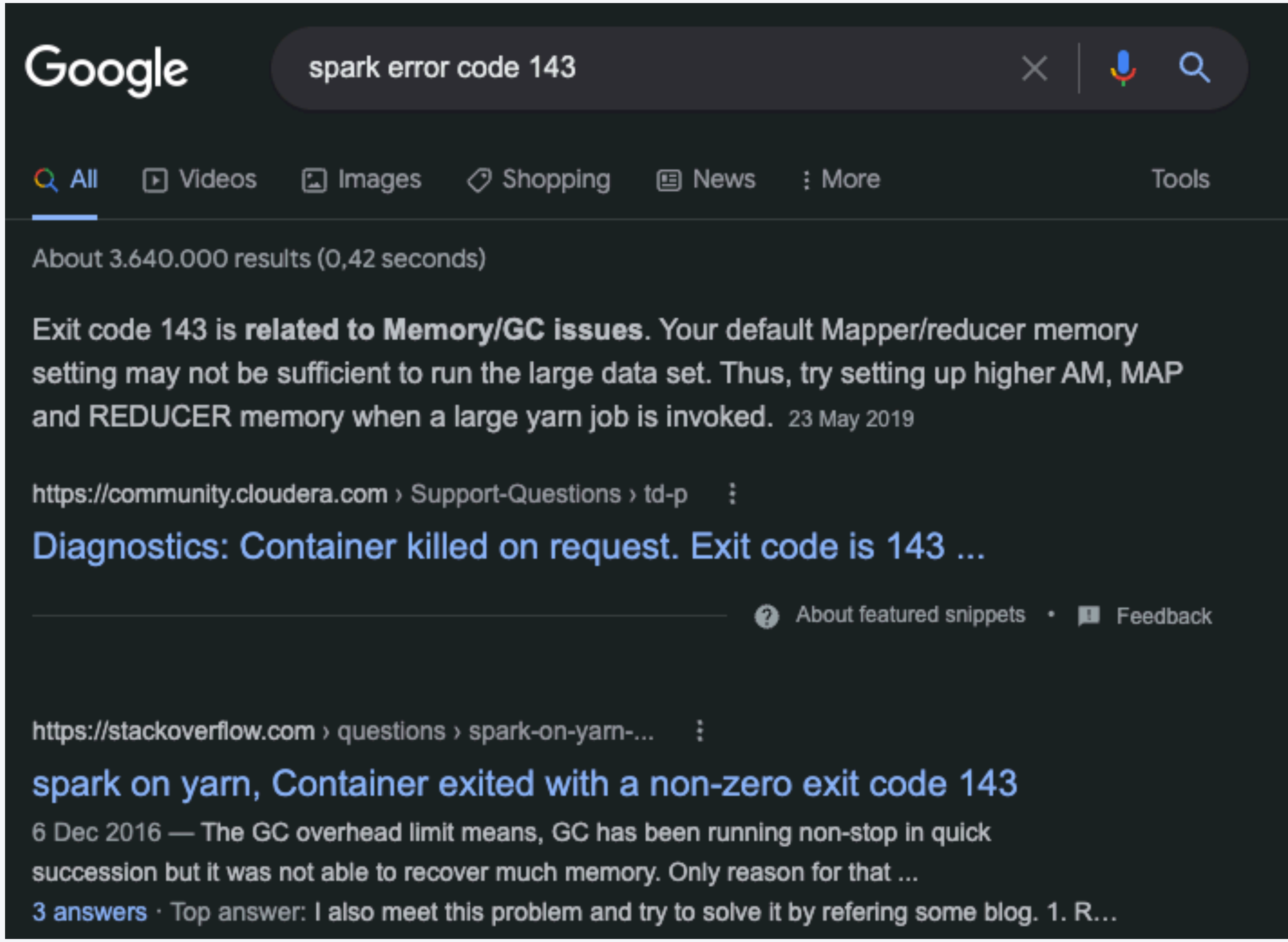
Tuning Resources

- As we are on-premise, we have finite resources
- We hardcode the resources in our code base
- Now imagine:
  - A DAG that has been running for 100 days
  - On day 101 you get ***error code 143***
  - What do you do?



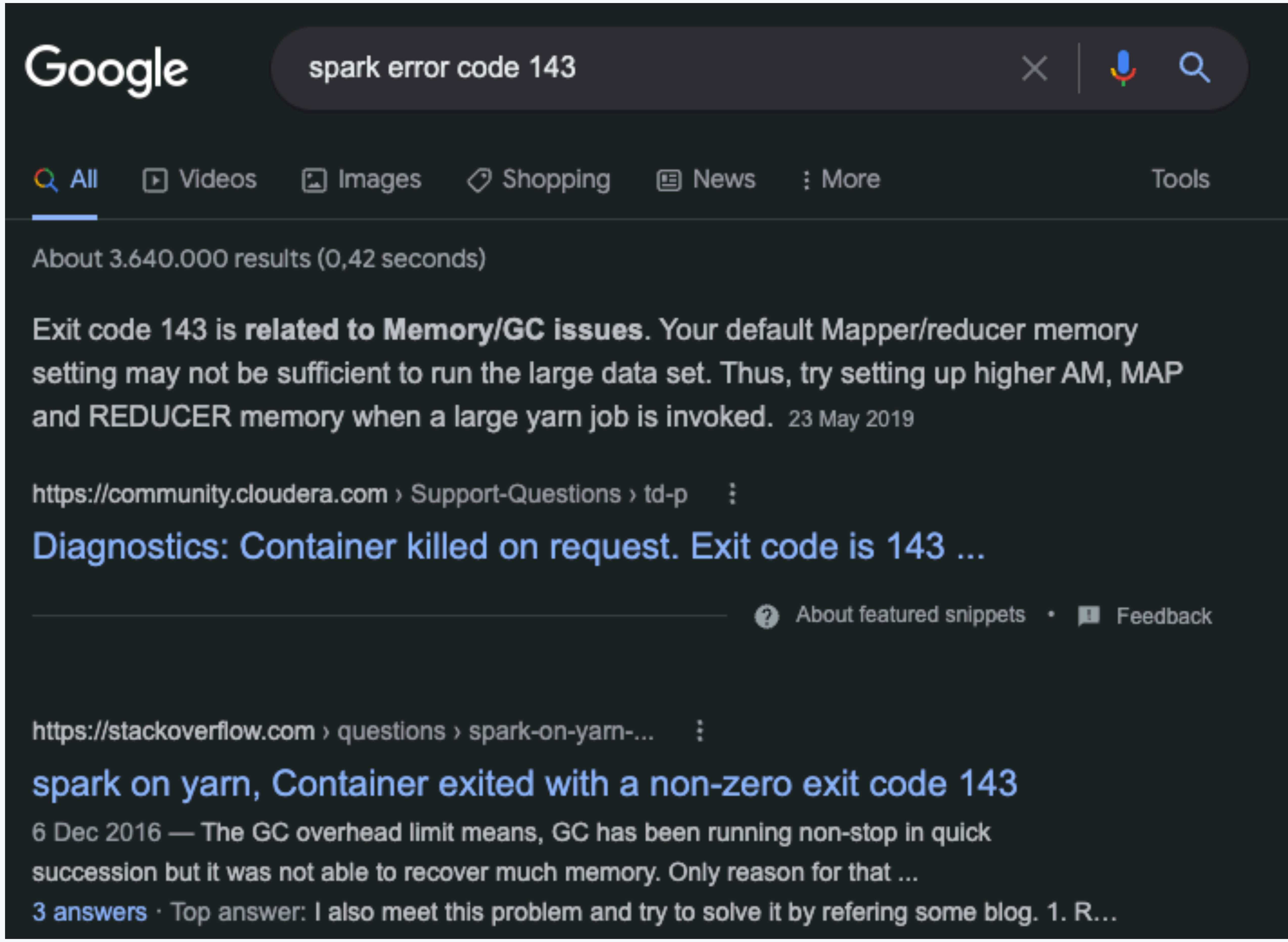
The Usual Workflow

- You google the issue
- Solution: Set a higher Spark driver and/or worker memory limit



The Usual Workflow

- You google the issue
- Solution: Set a higher Spark driver and/or worker memory limit
- But we hardcoded our resources in the code base, so that would require:
  - Create an MR
  - Request for an approval
  - Creating an official patch request
  - Request for an approval
  - Patching
- How can we prevent patching?



not-so-live demo


localhost
spark_daily_append - Tree - Airflow
Airflow
DAGs
Security
Browse
Admin
Docs
Duty
Cluster: dev-local
10:50 UTC
AA
DAG: spark_daily_append
Schedule: 0 2 * * *
Next Run: None
Tree
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
DAG Docs
2021-01-12T01:00:00Z
Runs
25
Update
ETLSourceOperator
OverwriteTableSensor
queued
running
success
failed
up_for_retry
up_for_reschedule
upstream_failed
skipped
scheduled
deferred
no_status
Auto-refresh
Jan 01, 02:00
Jan 07, 02:00
Jan 12, 02:00
[DAG]
sensor_some_other_dag.some_external_task
run_spark_ingestion_job

APACHE Spark™

DASK

druid

Retention Period

- As we are on-premise, we have finite storage.
- If we are close to storage limits, there are two options:
  - Buy more servers
  - Get rid of some of our data

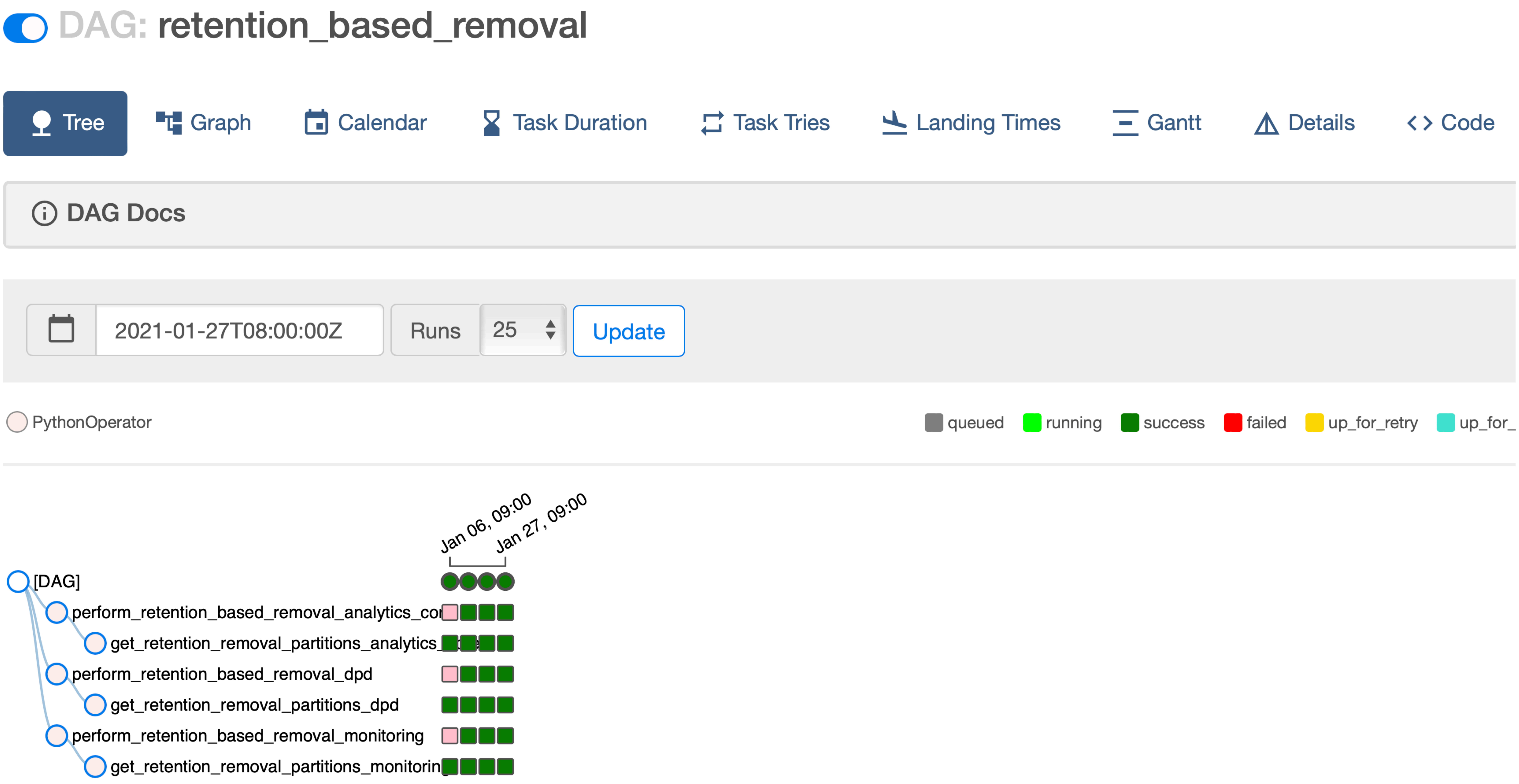
- We should prevent getting close to the limit

Governance library

- Each stream has a Stream file in this library.
- They define all their tables there and their corresponding retention period.
- Disclaimer: This is much more complex when you work with tables that are not partitioned by date

```python
class AnalyticsCore(Stream):
    def get_all_tables(self) -> List[HDFSTable]:
        return [
            HDFSTable(
              HiveDatabase.ANALYTICS_CORE, "parquetfilesize", RetentionPeriod.THREE_MONTHS
            ),
            HDFSTable(
              HiveDatabase.ANALYTICS_CORE, "parquetmetadatarowcount", RetentionPeriod.ONE_MONTH
            ),
            HDFSTable(
              HiveDatabase.ANALYTICS_CORE, "sparkpushedfilters", RetentionPeriod.ONE_YEAR
            ),
            HDFSTable(
              HiveDatabase.ANALYTICS_CORE, "tablemetadata", RetentionPeriod.THREE_YEARS
            ),
        ]

```


DAG: retention_based_removal
Tree
Graph
Calendar
Task Duration
Task Tries
Landing Times
Gantt
Details
Code
DAG Docs
2021-01-27T08:00:00Z
Runs
25
Update
PythonOperator
queued
running
success
failed
up_for_retry
up_for_
Jan 06, 09:00
Jan 27, 09:00
[DAG]
perform_retention_based_removal_analytics_co
get_retention_removal_partitions_analytics_
perform_retention_based_removal_dpd
get_retention_removal_partitions_dpd
perform_retention_based_removal_monitoring
get_retention_removal_partitions_monitorin

List Retention Removal Plan Model

Search ▾

\+ Actions ▾ ←

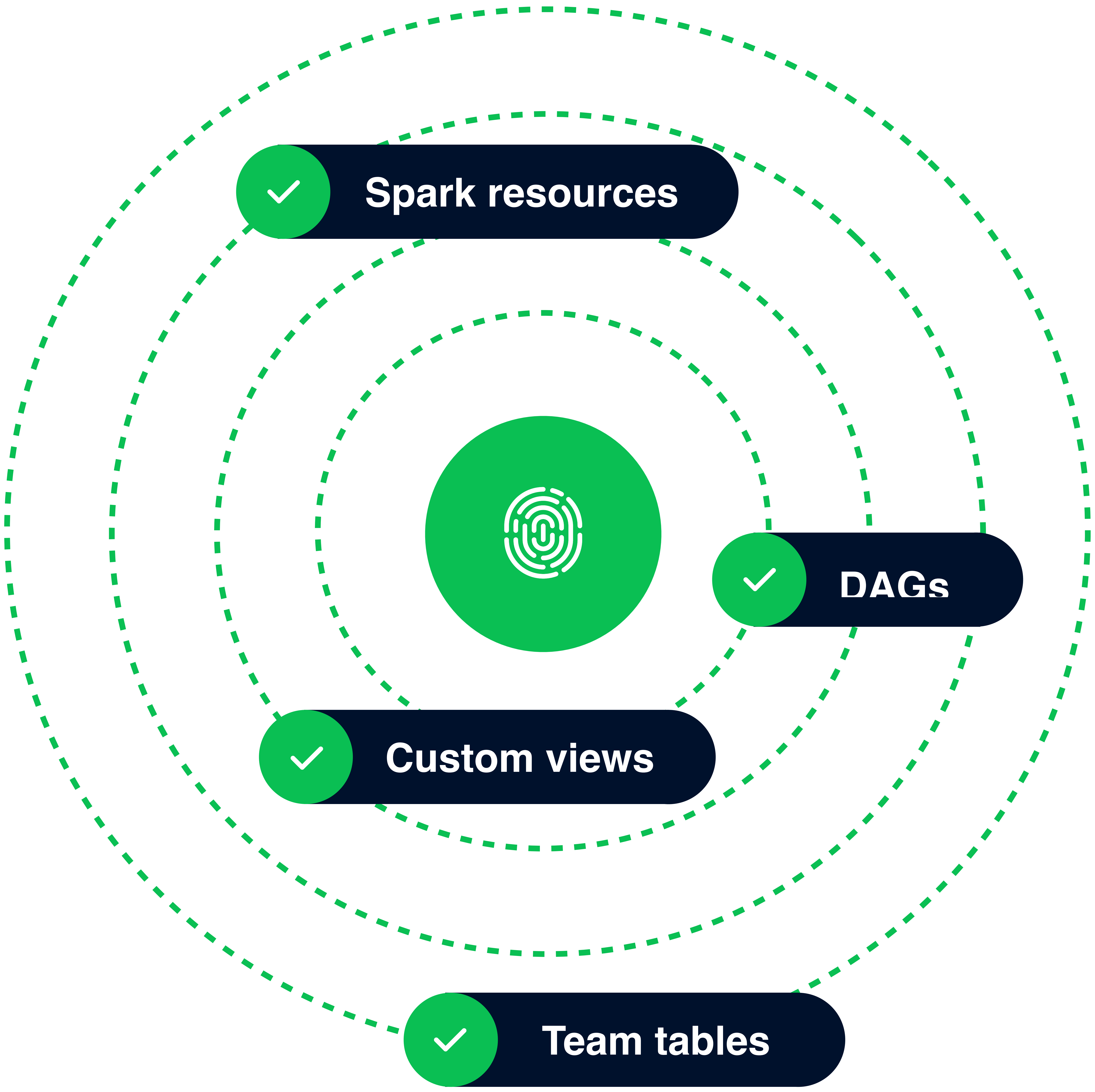
Record Count: 4

		Stream	Hive Database	Table Name	Planned Removal Date	Status	Partitions Start Date	Partitions End Date	Forbid Undo Dag
		analytics_core	analytics_core	sparkpushedfilters	2022-05-18	SUCCEEDED	2021-05-11	2021-05-17	False
		analytics_core	analytics_core	sparkpushedfilters	2022-05-25	STAGED	2021-05-18	2021-05-24	False
		analytics_core	analytics_core	tablemetadata	2022-05-18	SUCCEEDED	2022-05-05	2022-05-11	False
		analytics_core	analytics_core	tablemetadata	2022-05-25	STAGED	2022-05-12	2022-05-18	False

			e	Table Name	Planned Removal Date	Status	Partitions Start Date	Partitions End Date	Forbid Undo Dag
				sparkpushedfilters	2022-05-18	SUCCEEDED	2021-05-11	2021-05-17	False
		analytics_core	analytics_core	sparkpushedfilters	2022-05-25	STAGED	2021-05-18	2021-05-24	False
		analytics_core	analytics_core	tablemetadata	2022-05-18	SUCCEEDED	2022-05-05	2022-05-11	False
		analytics_core	analytics_core	tablemetadata	2022-05-25	STAGED	2022-05-12	2022-05-18	False

User Permissions

- We want to empower the streams so it truly becomes self-service
- Yet, they should only be able to modify their own resources.

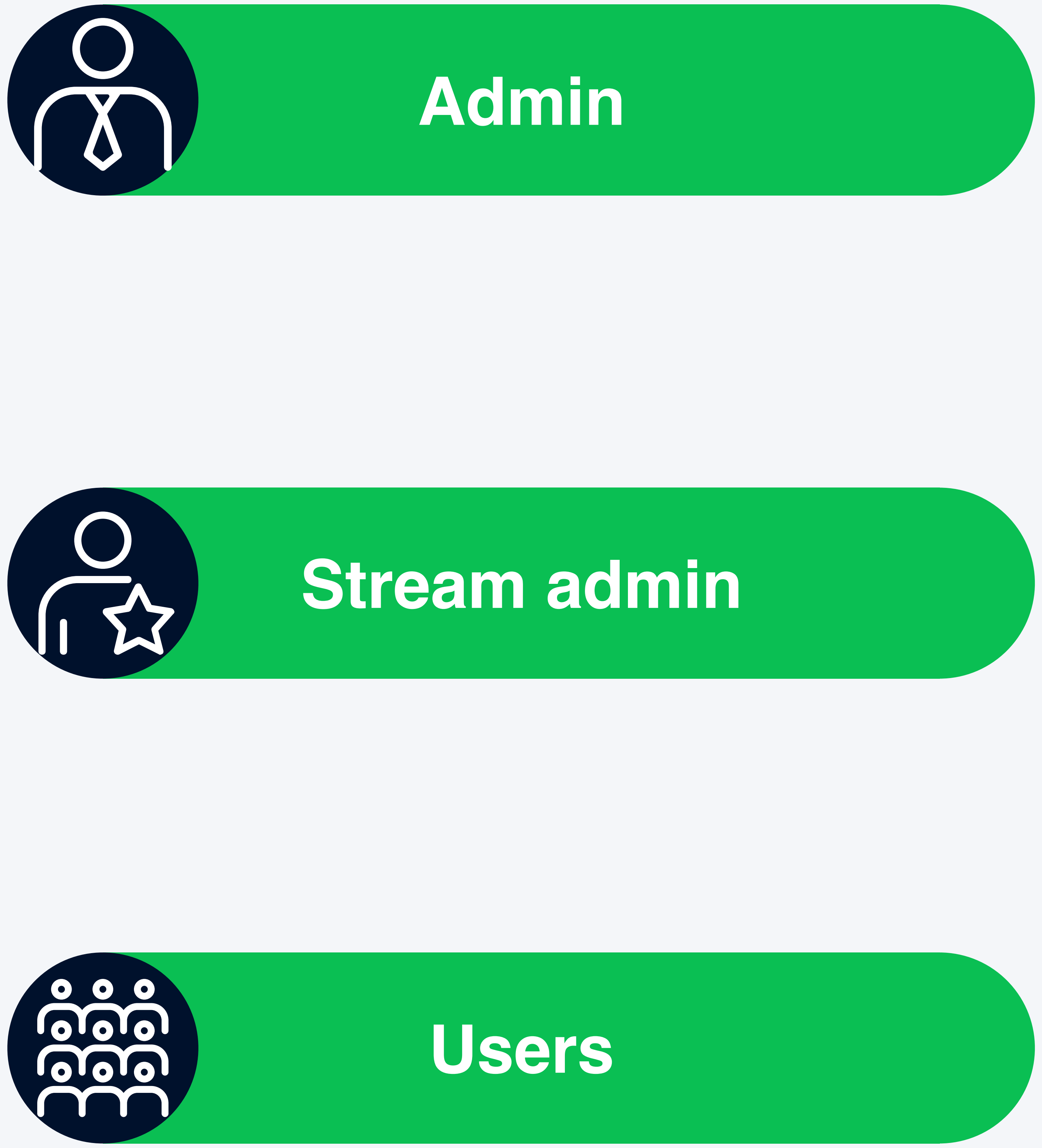


User Groups

- We always have one on-duty Admin

- For teams we have two access groups:
  - Stream admin
  - Standard user

- Team admins are able to:
  - Manage all the DAGs of the teams
  - Manage all the tables of their database
  - Manage the Spark resources of their tasks



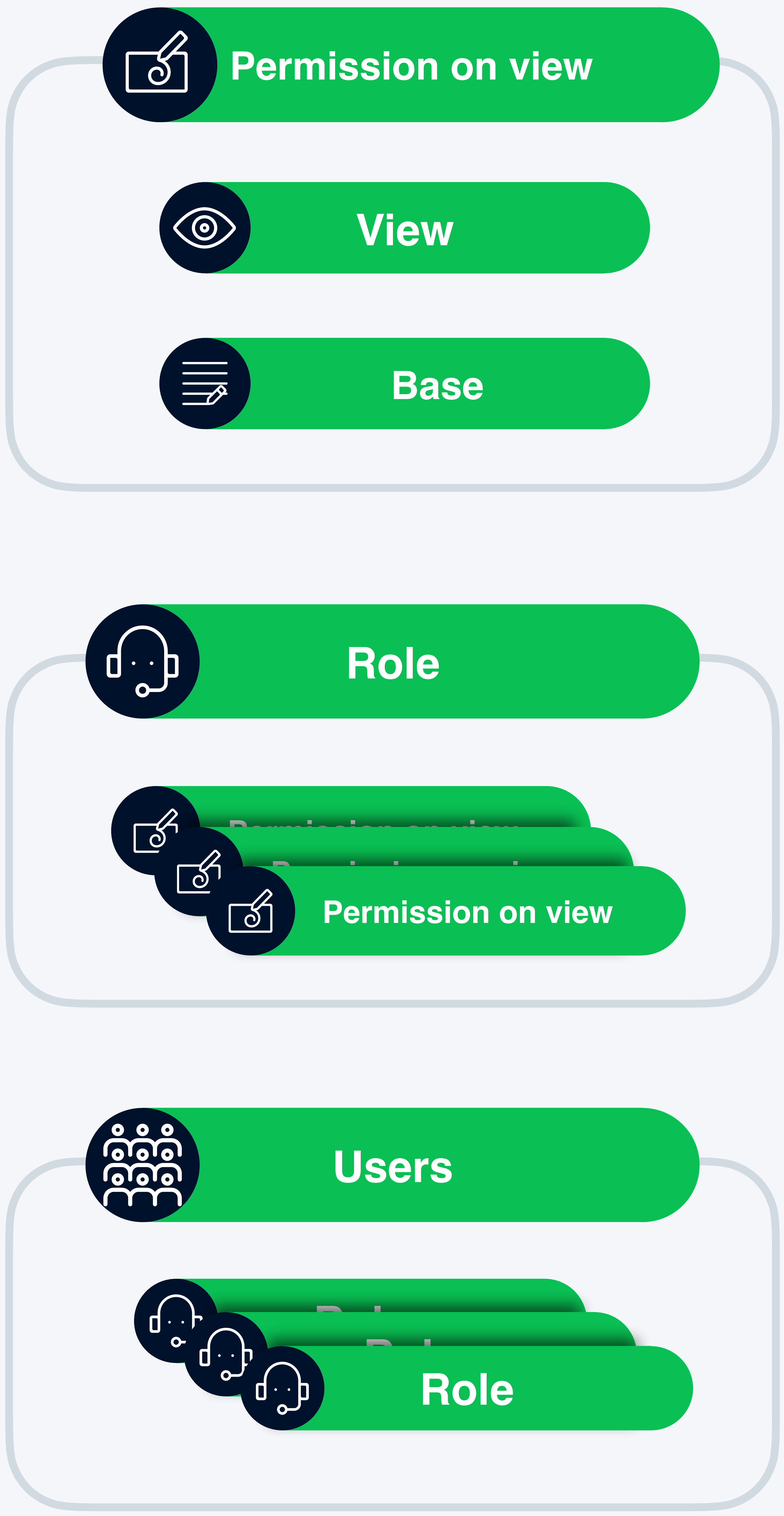
Flask Appbuilder Permissions

- Why are you talking about Flask Appbuilder?

- In terms of permissions on these views:
  - Each DAG has two POVs:
    - my_dag.can_read
    - my_dag.can_edit

  - Each function in a view has a POV:
    - spark_configuration.can_add
    - spark_configuration.can_edit

- For roles we create:
  - One basic role for all users
  - One role for each stream admin group



We Implemented

- On each release, all permissions per role are updated.
- Created decorators to indicate who has access.
- Validators that raise exceptions when someone tries to modify a task that it has no access on.
- Modified our views to only show runnable parts.

Admin

Stream admin

Users

not-so-live demo

Private

localhost

Airflow DAGs Security Browse Admin Docs Duty Cluster: dev-local 13:28 UTC AA

List Users

Search

Record Count: 2

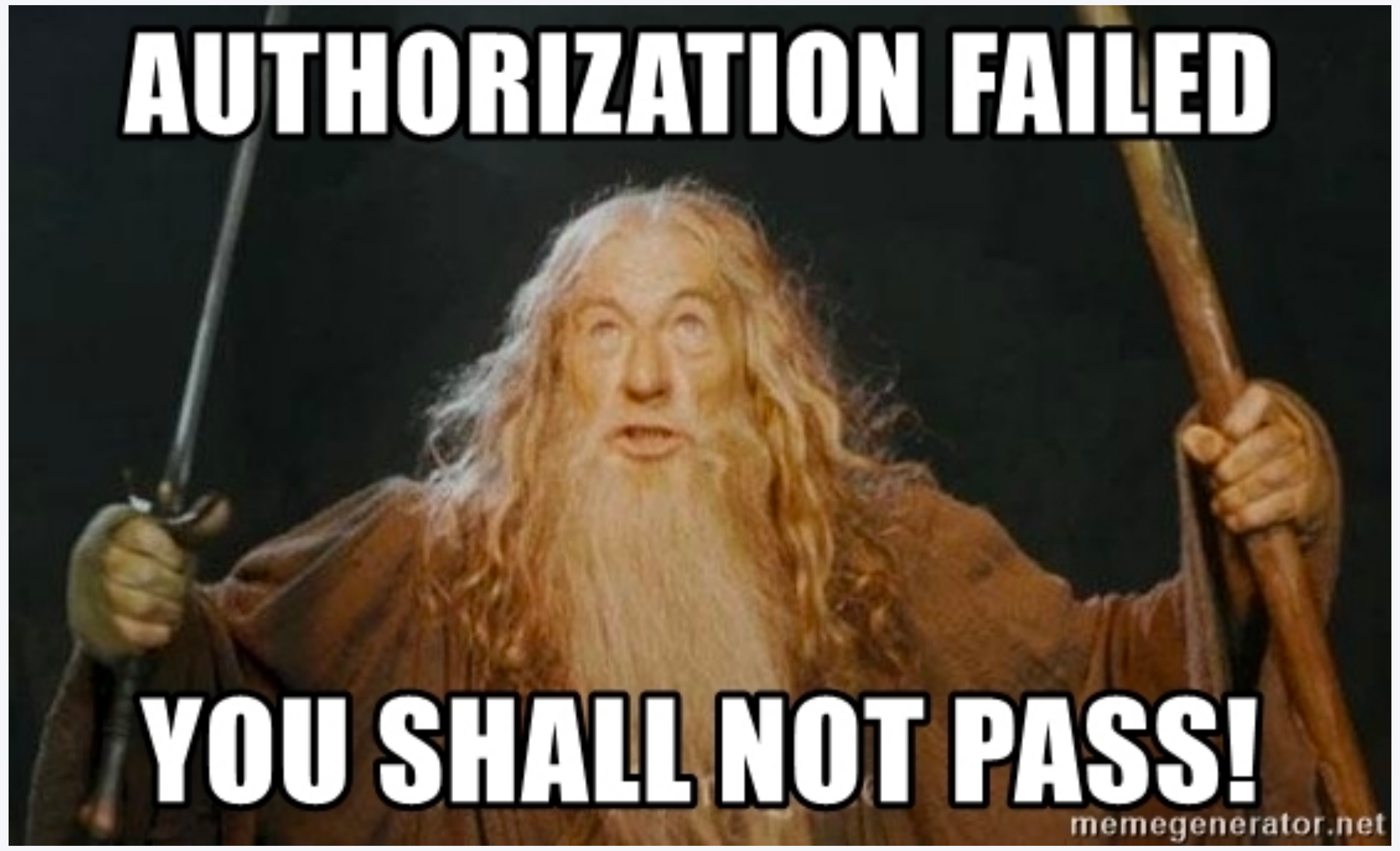
	First Name	Last Name	User Name	Email	Is Active?	Role
	ac	ac	ac	ac@adyen.com	True	[User, stream_analytics_core]
	adyen	adyen	adyen	adyen@adyen	True	[Admin]

Version: v2.2.3

Git Version: .release:2.2.3+06c82e17e9d7f1bf261357e64c6013ccdb3c241

Mapping Users to Roles

- Now that we have the roles defined, this is the final thing left to do.
- Security team requires us to use LDAP for managing the roles.
- Service that runs each hour.



Wrap up

Table Schemas

How one can enable schema evolution over time and abstract away complexities

Tuning Resources

Preventing patches for data overflowing its ETL resource limits

Retention Period

Work with limited storage

User Permissions

Each domain has their own database.

Before I forget…

NL - Amsterdam
Head office

US - Chicago
Tech hub

Spain - Madrid
Tech hub

Thank you

Slides for questions

Code Example: How we Define Permissions

```python
class ConfigurationView(AdyenProtectedModelView):
    datamodel = CustomSQLAInterface(ConfigModel)

    extra_allowed_user_functions = ["list"]
    extra_allowed_stream_admin_functions = ["add", "delete", "edit"]

    @action("export", "Export the Spark Configurations", "", single=False)
    @visible_to_users(is_fab_action=True, action_name="export")
    def export_configs(self, configs: List[ConfigModel]) -> Response:
        pass

    @action("drop", "Drop Configurations completely", "You sure?", single=False)
    @visible_to_stream_admins(is_fab_action=True, action_name="drop")
    def drop_completely(self, configs: List[ConfigModel]) -> Response:
        pass
```

Code example: Implementation of Decorators

```python
def visible_to_stream_admins(is_fab_action: bool, action_name: Optional[str] = None):
    def inner_function(func: Callable):
        @wraps(func)
        def wrapper(*args: Any, **kwargs: Any) -> Any:
            return func(*args, **kwargs)

        if is_fab_action and (action_name is None or not any(action_name)):
            raise ValueError(f"If {is_fab_action=} you must also set {action_name=}.")

        wrapper.visible_to_stream_admins = True
        wrapper.is_fab_action = is_fab_action
        wrapper.fab_action_name = action_name
        return wrapper

    return inner_function
```

Operations on Tables

For any operation you wish to perform on a table, you can categorise them in one (or multiple) of these 5 operations:

	NEW	COMPATIBLE	INCOMPATIBLE	REMOVE	MOVE
Defines hive table definition	☑	☑	☑		☑
Delete hive table definition		☑	☑	☑	☑
Delete table data	☑		☑	☑	
MSCK Repair		☑			☑