May 23–27, 2022

AIRFLOW SUMMIT

Airflow
Summit 2022

Well-Architected Workflows - Resiliency

Uma Ramadoss
Sr. Specialist Solutions Architect, Integration
Amazon Web Services

What is Resiliency?

Capability of the workload to recover from infrastructure or service disruptions

Reliability

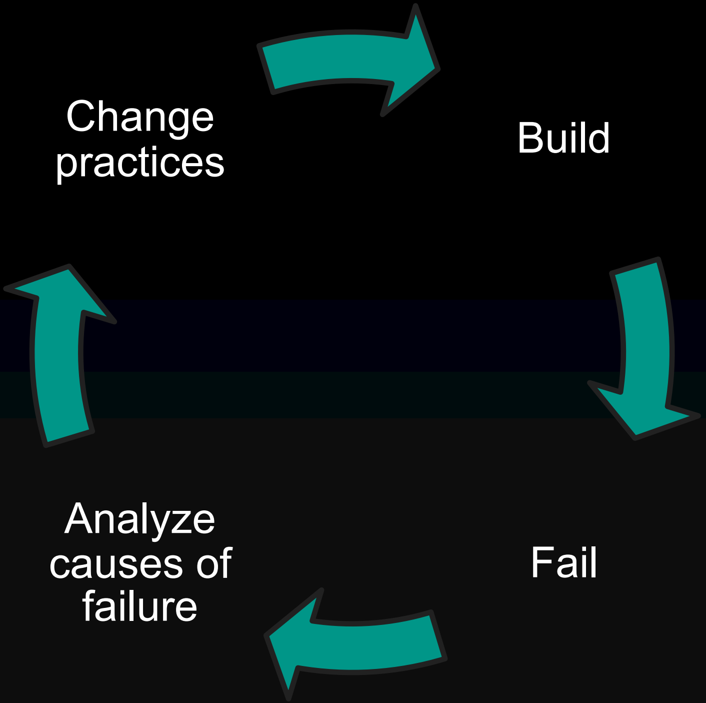
- Ability of the workload to perform its intended function correctly and consistently.
- Reliability is impacted by operational practices, performance efficiency, security etc. including Resiliency

Resiliency

- Ability of the system to recover from failures
- Resiliency is the component of Reliability

What are Resilient workflows?

Redriveable workflows with retriable atomic tasks.



Change practices

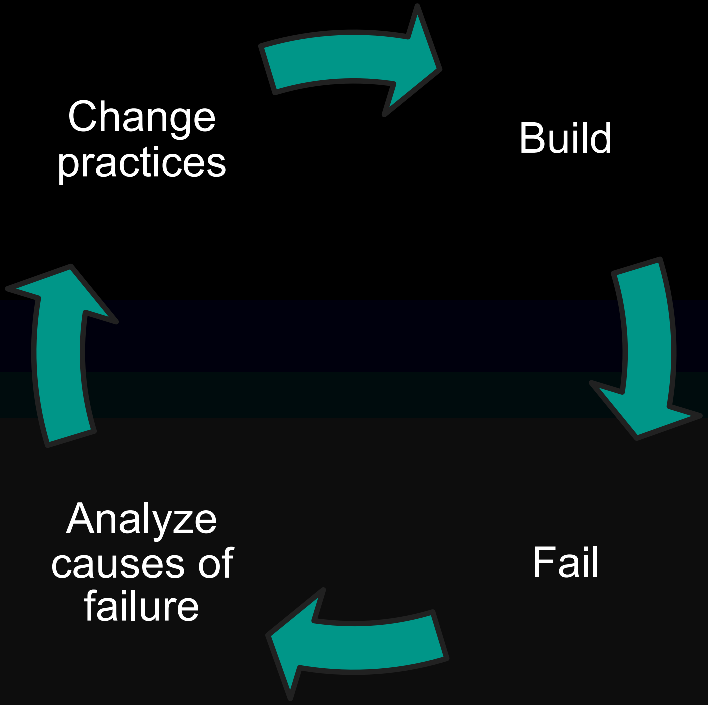
Build

Fail

Analyze causes of failure

> “Everything fails, all the time.”

Werner Vogels
CTO, amazon.com

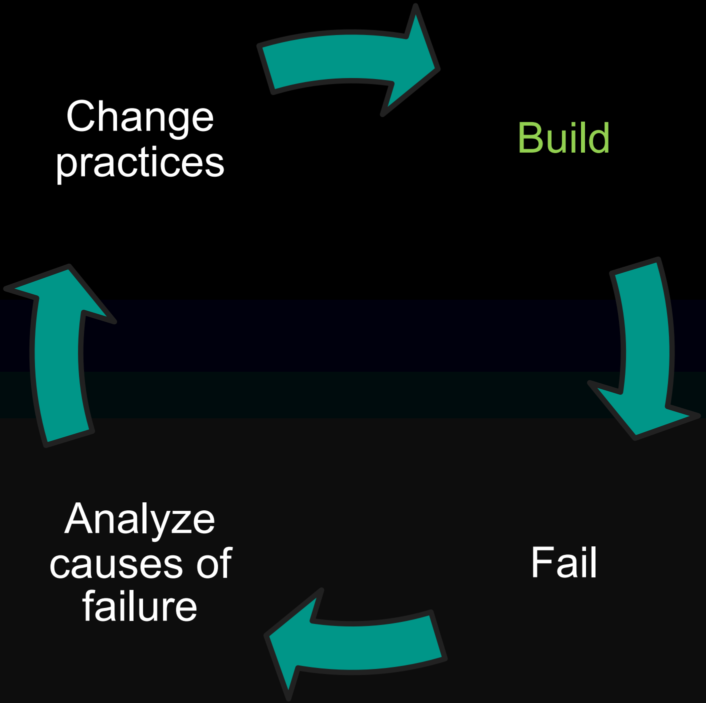


Change practices

Build

Fail

Analyze causes of failure



Change practices

Build

Fail

Analyze causes of failure

Resiliency in Airflow Architecture

Understanding Main Components of Apache Airflow

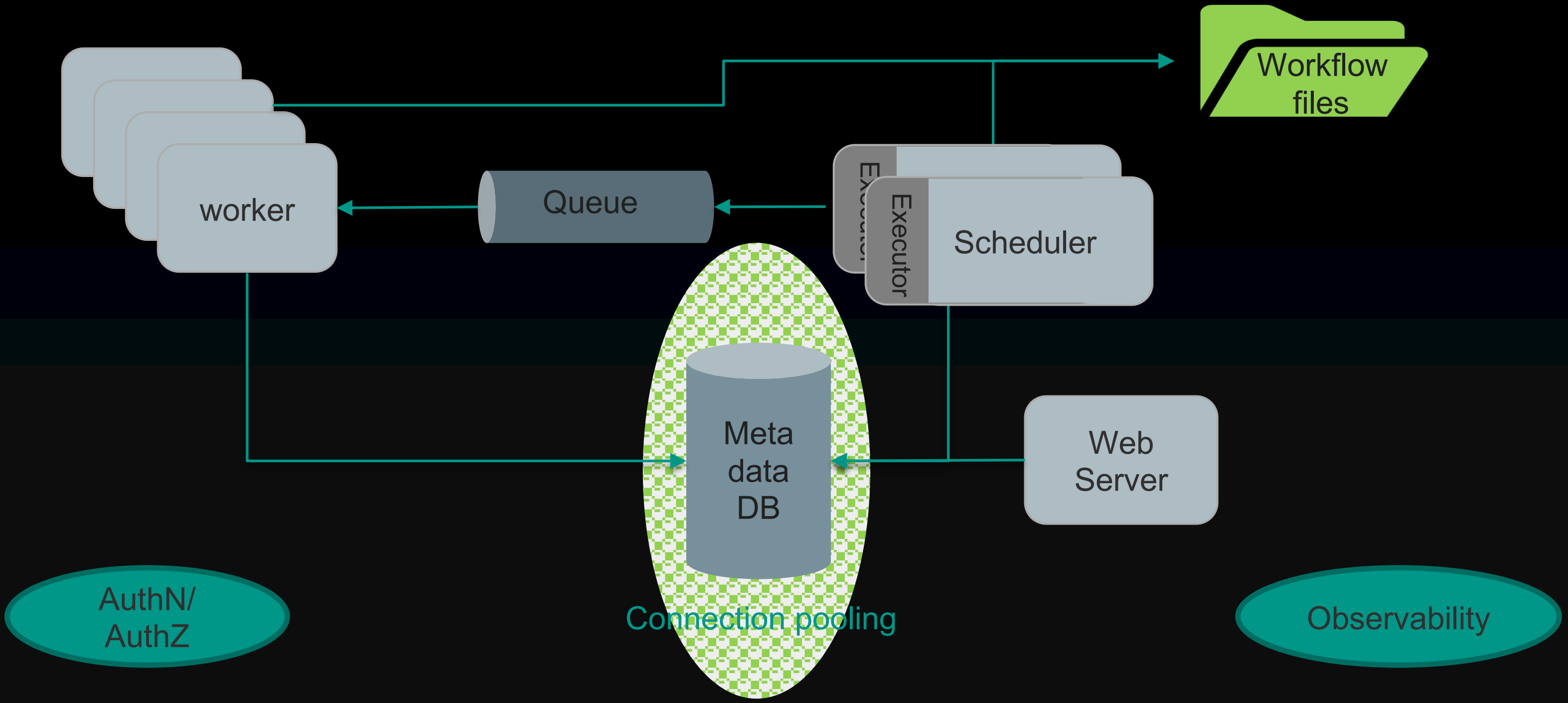
Scheduler

Worker

Web Server

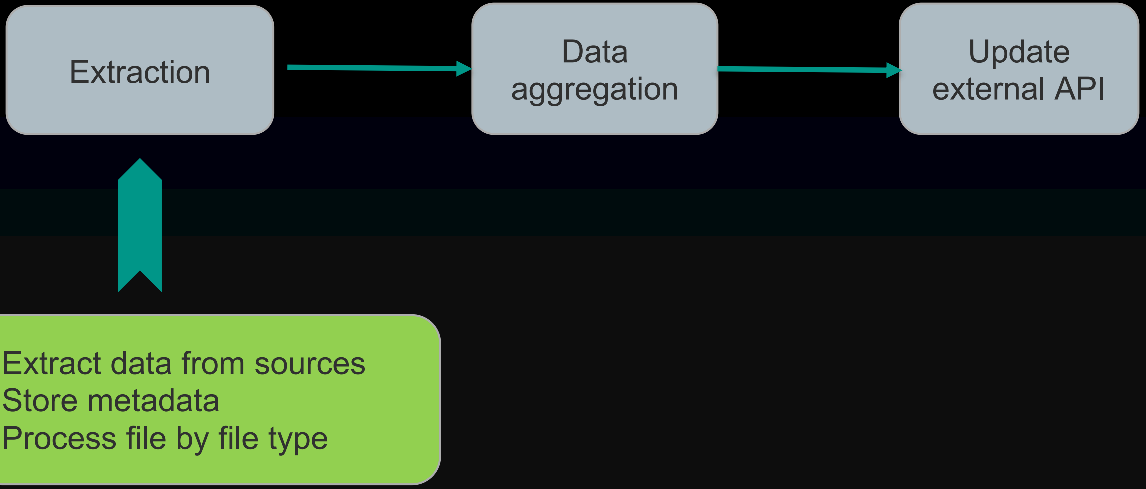
Meta Database

Production suitable implementation



Resilient Design Principles

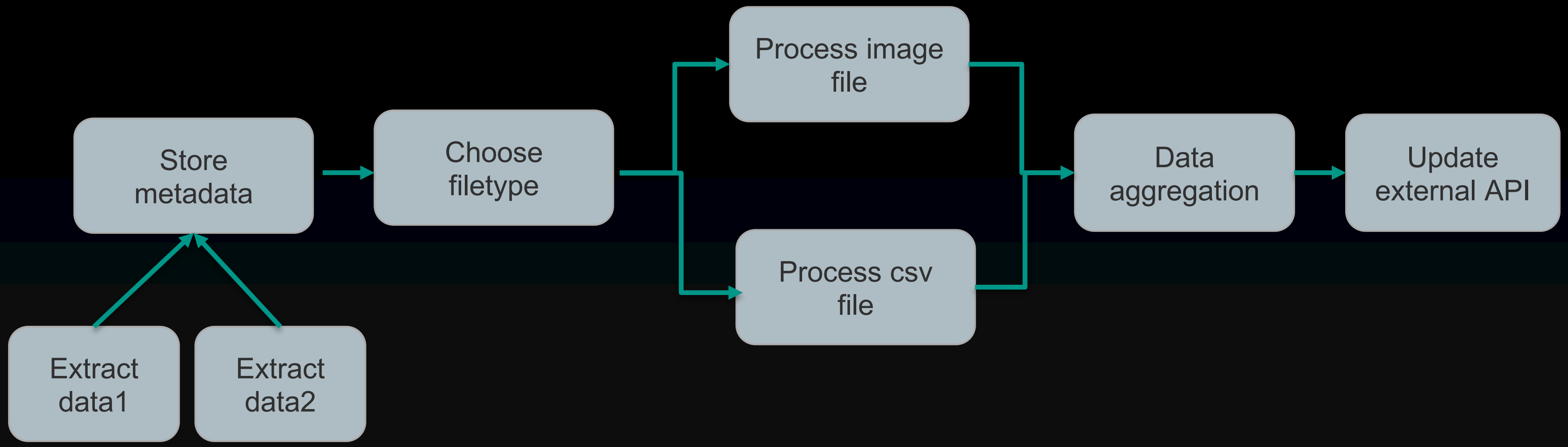
All or None - Atomicity



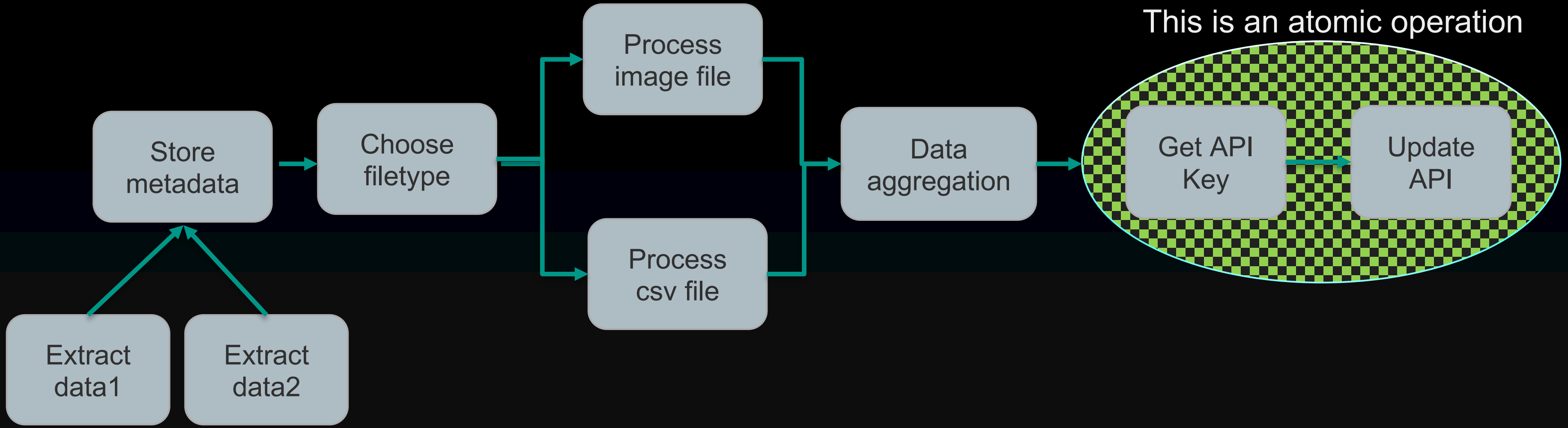
Extract data from sources
Store metadata
Process file by file type

All or None - Atomicity

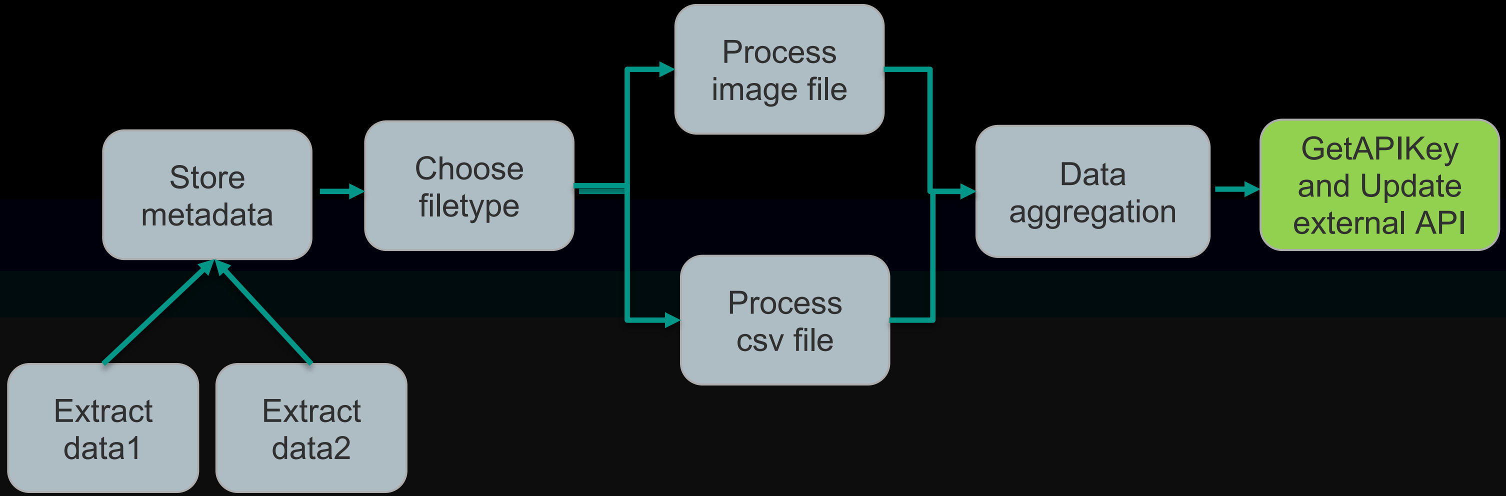
Airflow Summit 2022



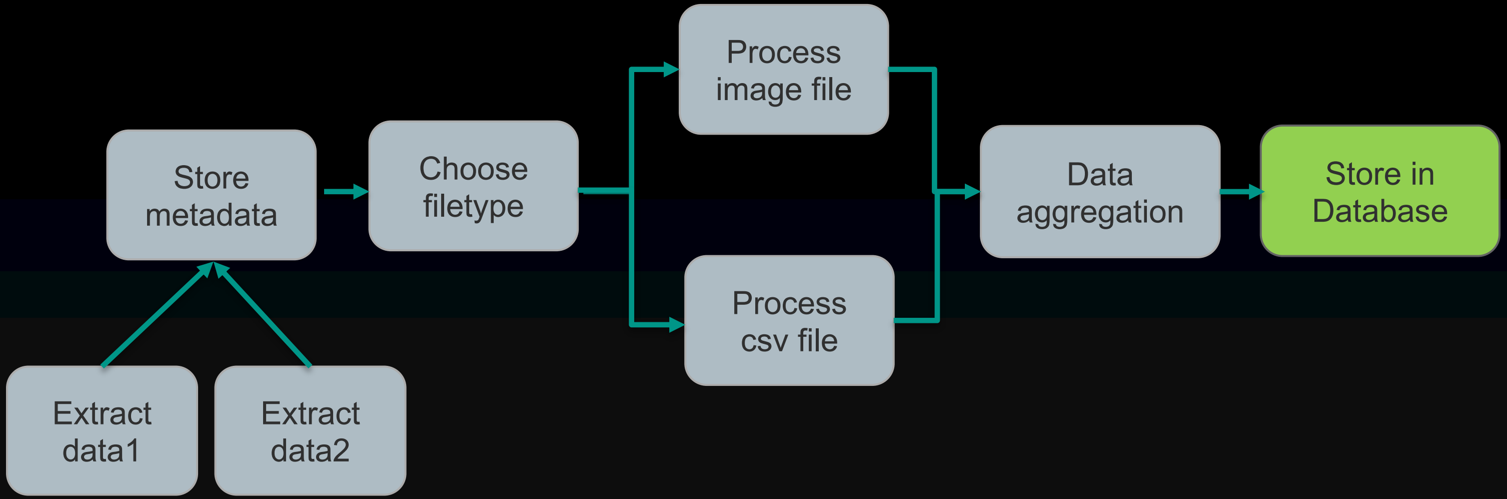
All or None - Atomicity



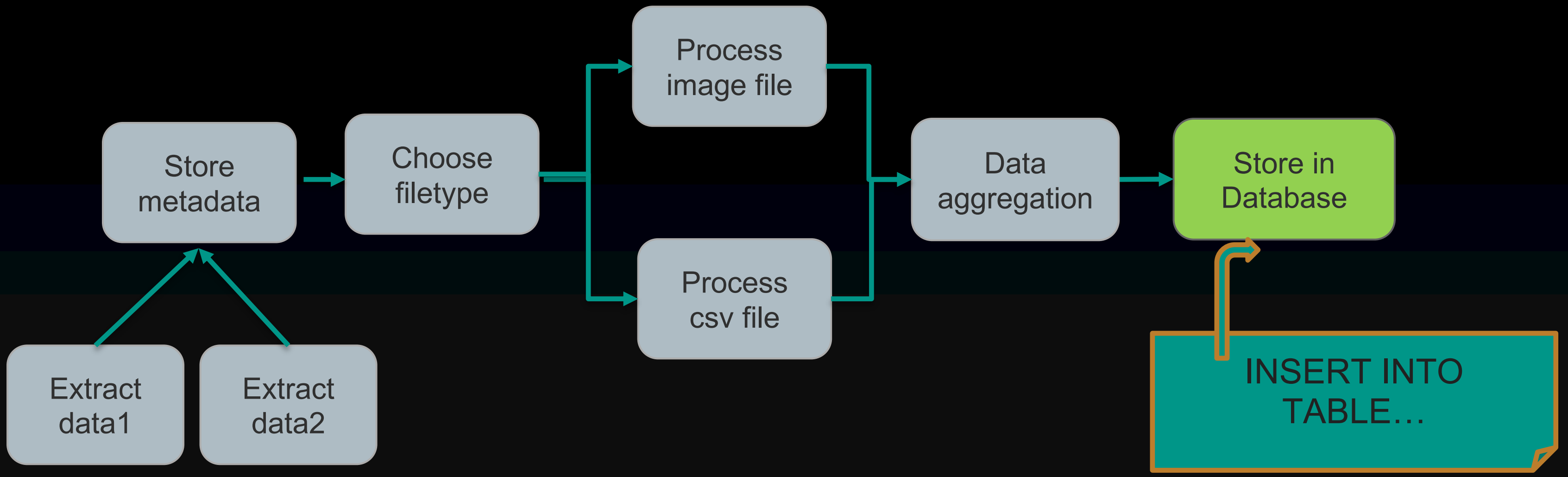
All or None - Atomicity



Make failures safe - Idempotency

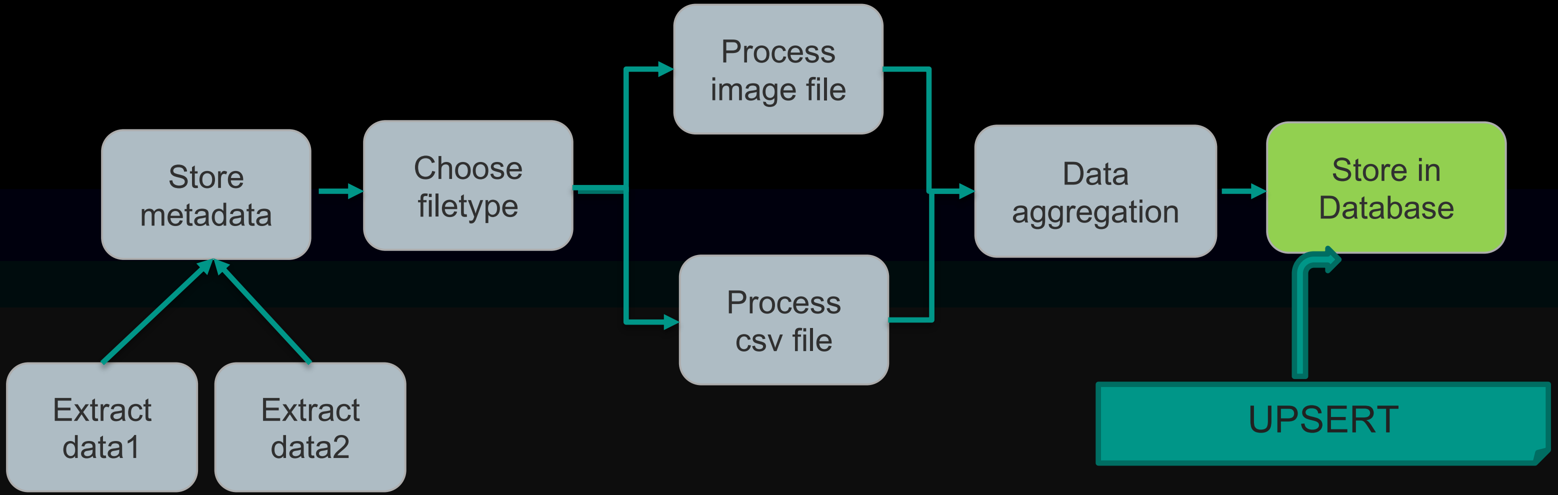


Make failures safe - Idempotency

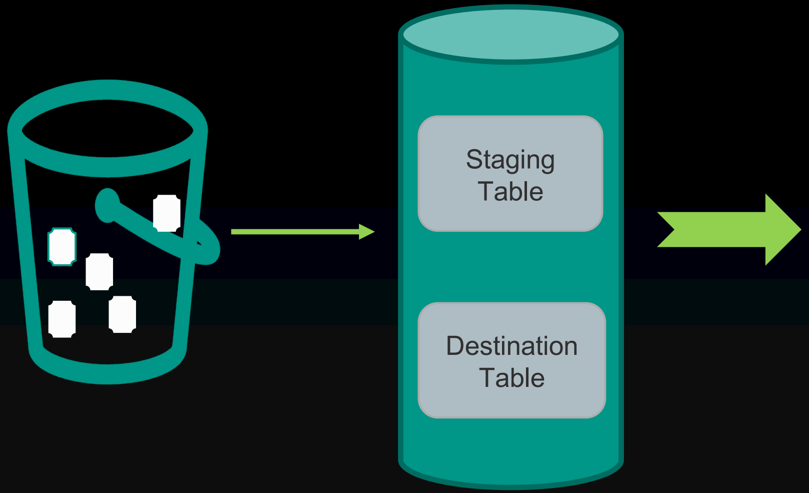


INSERT INTO TABLE…

Make failures safe - Idempotency

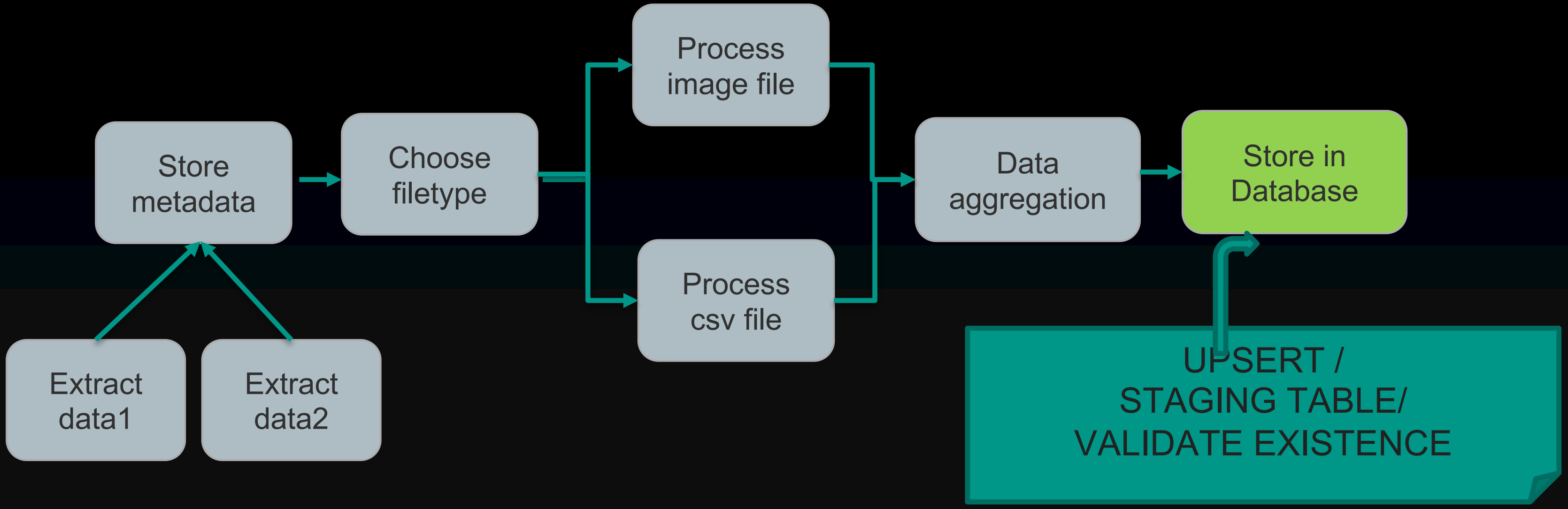


Make failures safe – Idempotency – Redshift UPSERT



```
task_transfer_s3_to_redshift =
        S3ToRedshiftOperator(
                s3_bucket=S3_BUCKET_NAME,
                s3_key=S3_KEY,
                schema='PUBLIC’,
                table=REDSHIFT_TABLE,
                copy_options=['csv’],
                method=‘UPSERT’,
                task_id='transfer_s3_to_redshift’,
        )
```

Make failures safe - Idempotency



Extract data1

Extract data2

Store metadata

Choose filetype

Process image file

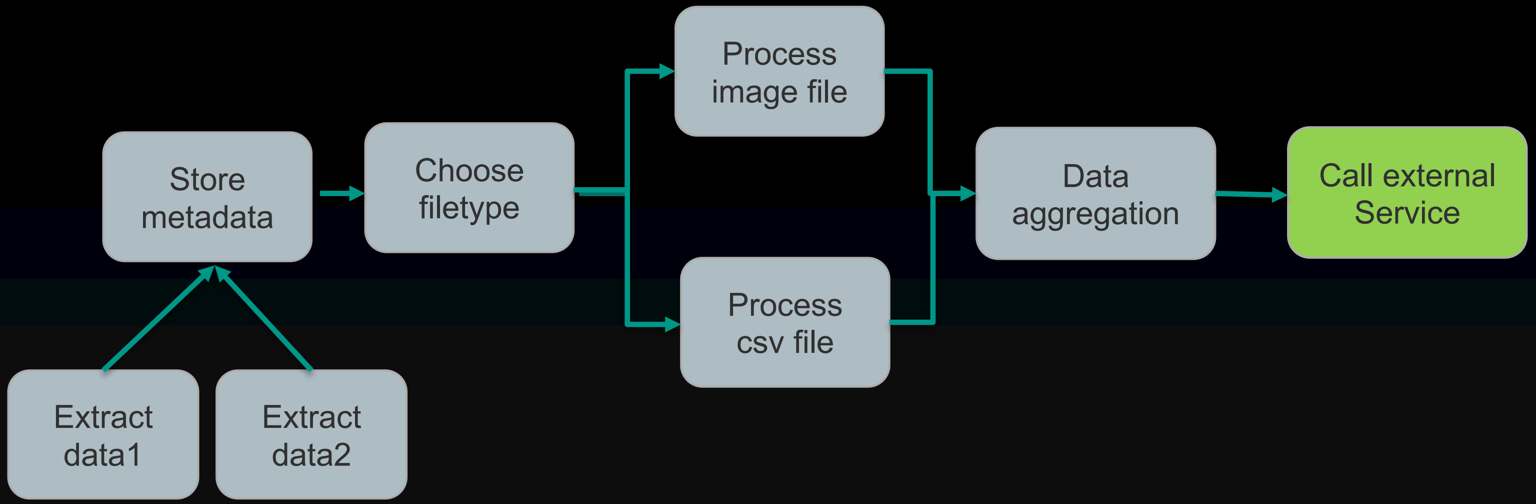
Process csv file

Data aggregation

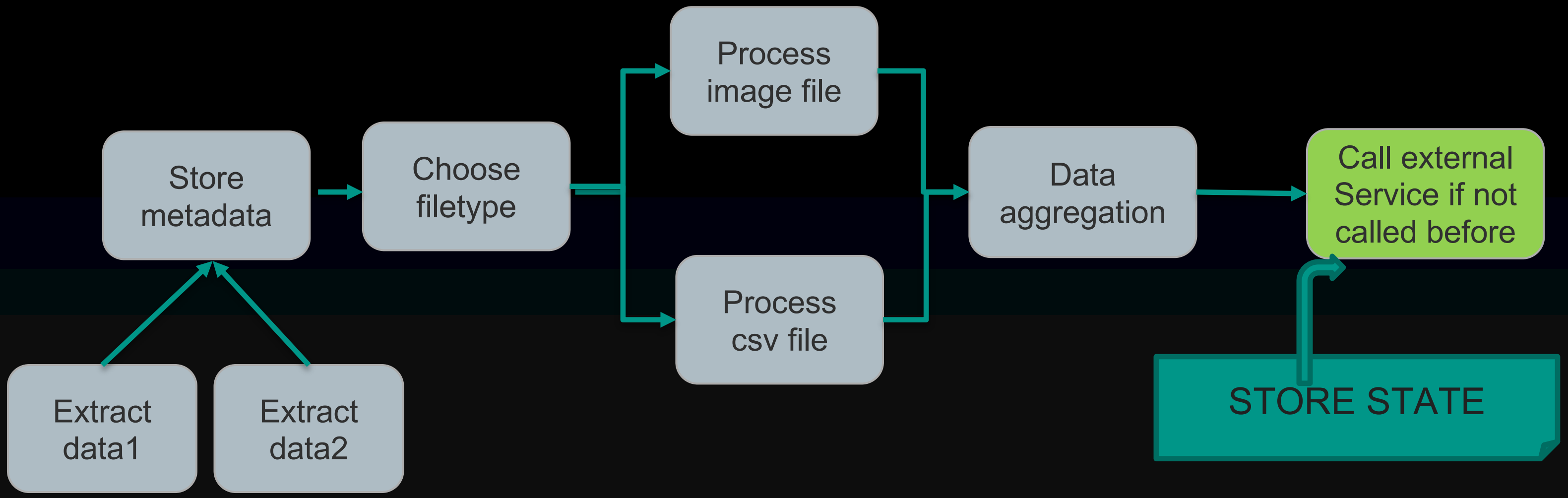
Store in Database

UPSERT /
STAGING TABLE/
VALIDATE EXISTENCE

Make failures safe - Idempotency



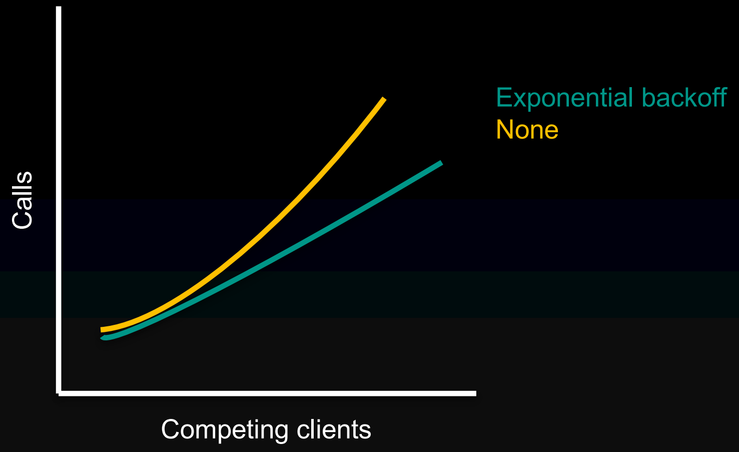
Make failures safe - Idempotency



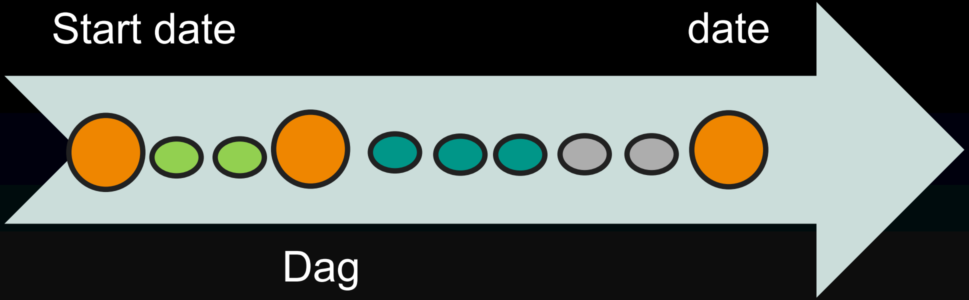
Protect downstream – retries, backoff

```
args=
{
  'depends_on_past': False,
  'email': ['airflow@example.com’],
  'email_on_failure’: True,
  'email_on_retry': False,
  'retries’: 3,
  'retry_delay': timedelta(seconds=5),
  ‘retry_exponential_backoff’: True

}
```



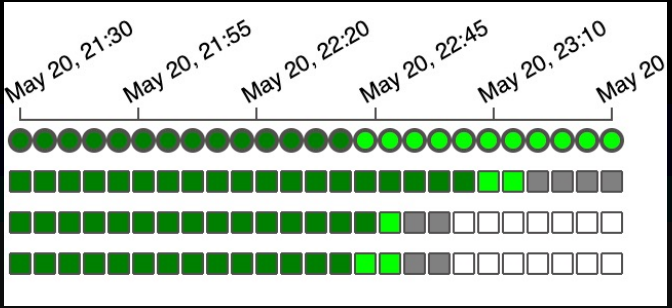
Protect downstream – max active runs



Backfill
Start date

Current
date

Dag
Start date



Protect downstream – Preemptive load shedding

Airflow pools can be used to **limit the execution parallelism** on arbitrary sets of tasks

Typically this is done to limit downstream impact, for example putting all database tasks in an “RDS” pool that has a limit based upon the connection limit of the DB

Fail fast and fail forward – SLA and Timeout

- Leverage SLA and sla_miss_callback for awareness
- Use execution timeout for cancellation of tasks
- Raise AirflowSkipException, AirflowFailException to fail fast on obvious errors
- Checkpoint/validate data

```
@task(sla=timedelta(seconds=60),
      execution_timeout=timedelta(seconds=70)
def long_running_task():
  blah = call_external_service()
  if blah == “foo”:
    raise AirflowSkipException
  .....

Validate_data = SQLCheckOperator
                 (task_id=validate,…)
```

Resiliency Design Principles – Recap

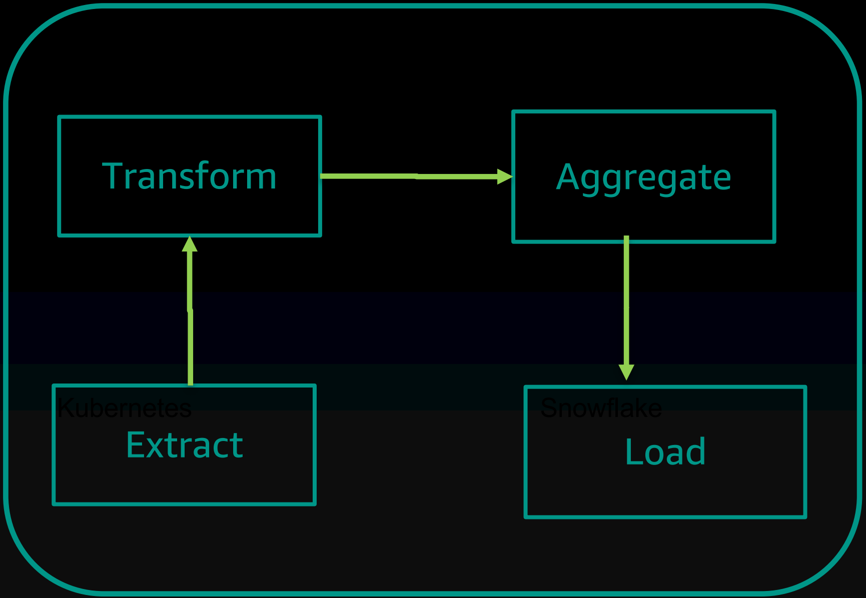
- All or None - Atomicity
- Make failures safe – Idempotency
- Protect Downstream
- Fail fast and fail forward

Some best practice implementations

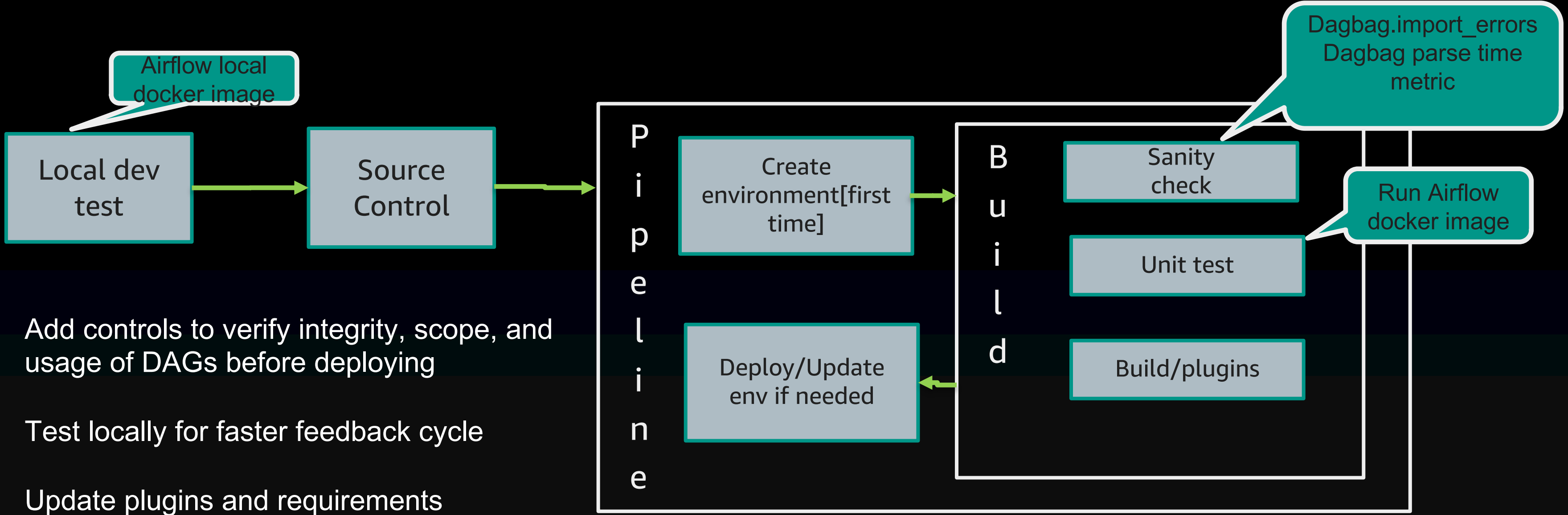
Use Airflow as an orchestration tool

Airflow Summit 2022

- Externalize compute/memory-intensive work to purpose built services.
- Leverage community offered operators.



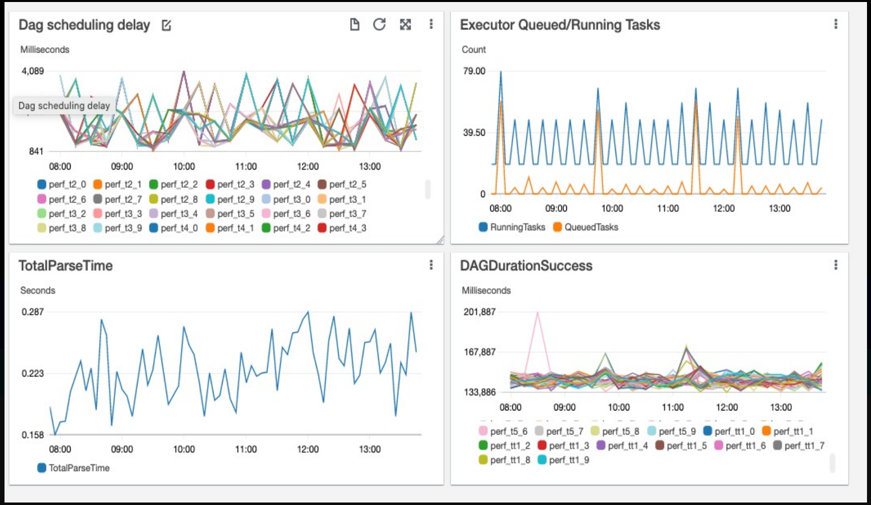
Operation as code & small reversible changes


Airflow local docker image
Local dev test
Source Control
Pipeline
Create environment[first time]
Deploy/Update env if needed
Build
Sanity check
Unit test
Build/plugins
Dagbag.import_errors Dagbag parse time metric
Run Airflow docker image

- Add controls to verify integrity, scope, and usage of DAGs before deploying
- Test locally for faster feedback cycle
- Update plugins and requirements programmatically

Monitoring Workflow

- Build dashboards with relevant metrics like parse time, scheduling delays, queued/running tasks etc
- Send notification when thresholds are exceeded.
- Leverage dashboards like Landing times, Gantt chart to troubleshoot performance issues



Testing in Airflow – Medium article

https://bit.ly/3w0PpeA

Data validation

https://www.youtube.com/watch?v=6ib2gH4A0rI

Airflow best practices

https://bit.ly/3LY1Hdh

Q/A

Thank you!

https://www.linkedin.com/in/umamaheswari-r-96578910/