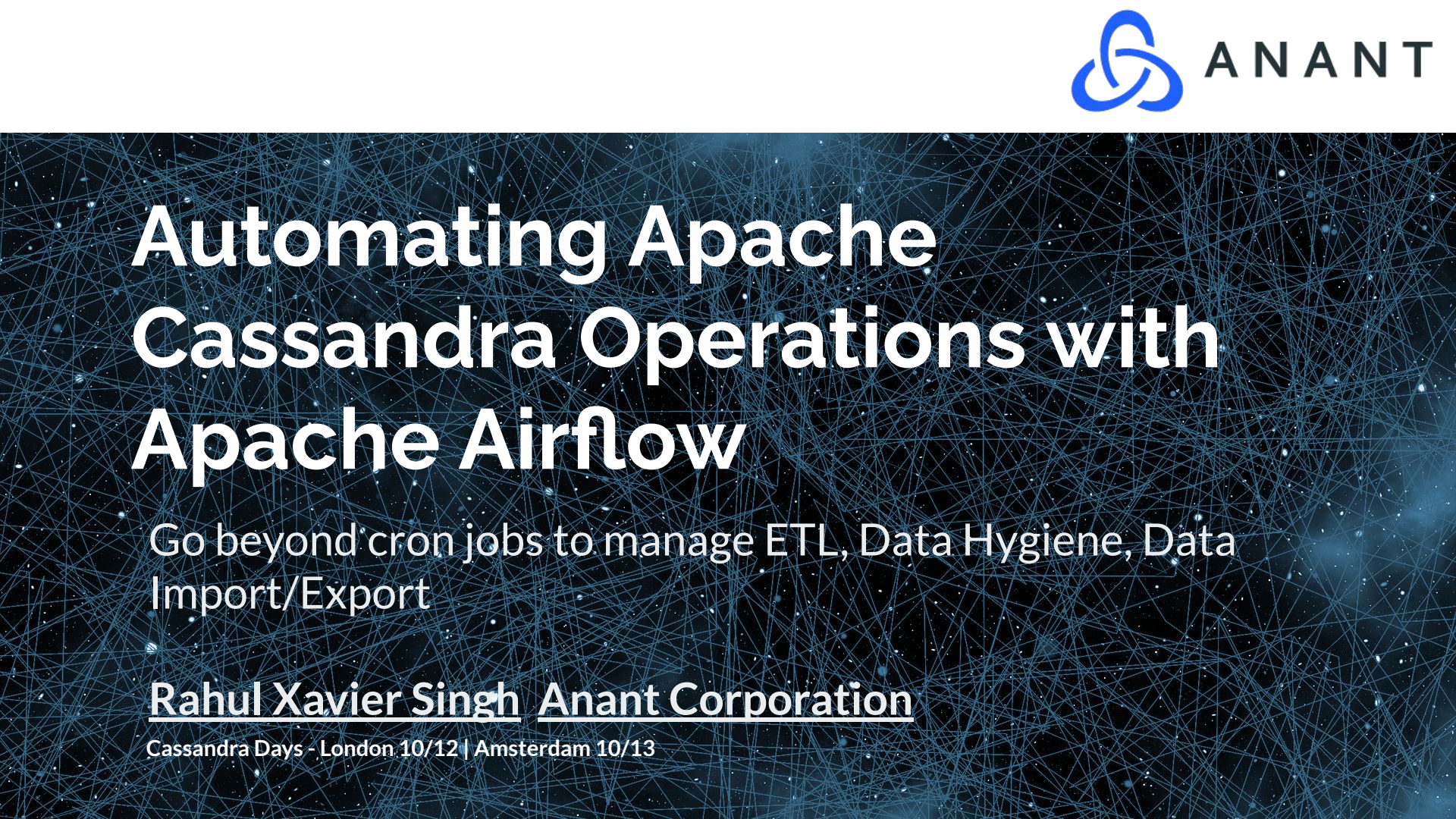

ANANT
Automating Apache Cassandra Operations with Apache Airflow
Go beyond cron jobs to manage ETL, Data Hygiene, Data Import/Export
Rahul Xavier Singh Anant Corporation
Cassandra Days - London 10/12 | Amsterdam 10/13

We help **platform owners** reach **beyond** their **potential** to serve a **global customer** base that demands

Everything, Now.

ANANT

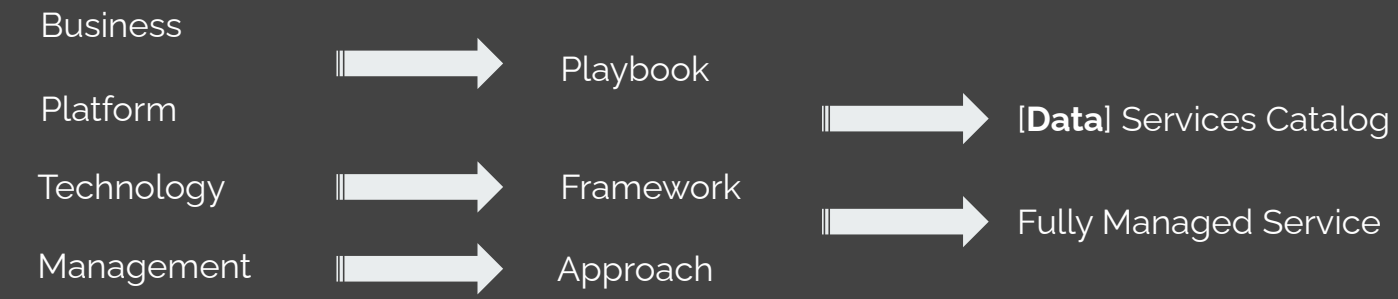
We design with our **Playbook**, build with our **Framework**, and manage platforms with our **Approach** so our clients

Think & Grow Big.

ANANT

Customer Success

Challenge Solutions Subscriptions



We offer Professional Services to engineer Solutions and offer Managed Services to clients where it makes sense, after an Assessment

ANANT

Business / Platform Dream

Enterprise Consciousness :

- People
- Processes,
- Information
- Systems

Connected / Synchronized.

Business has been chasing this dream for a while. As technologies improve, this becomes more accessible.

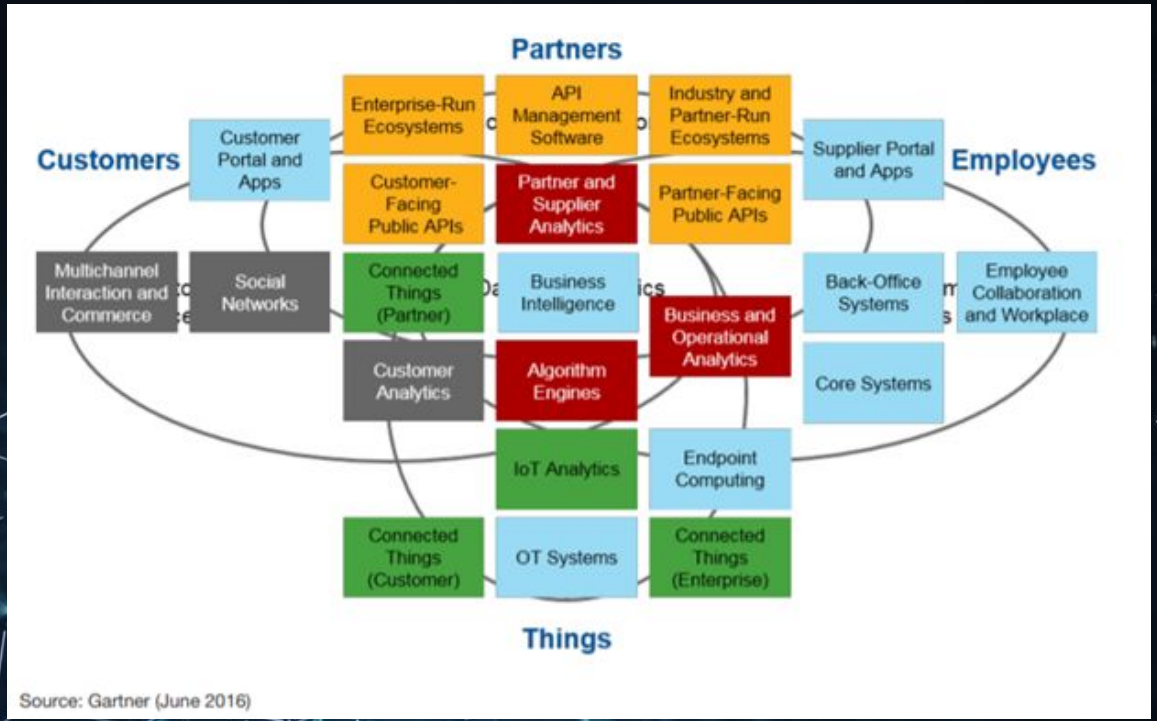
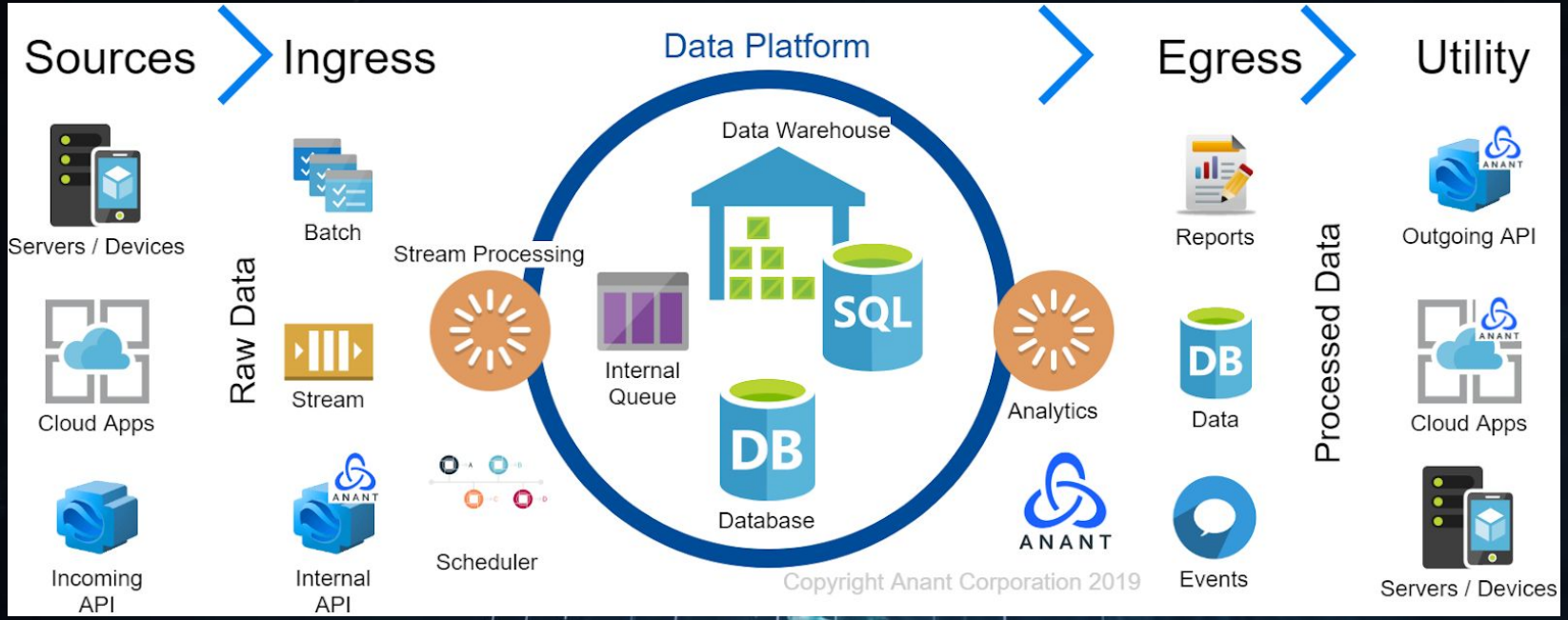


Image Source: Digital Business Technology Platforms, Gartner 2016

ANANT

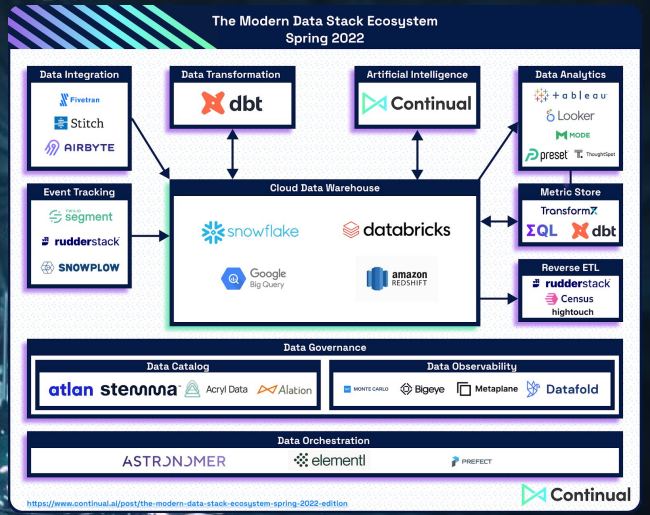
Modern Open Data Platform

Generic Data Platform Operations



How do you choose from the landscape?

So Many Different “Modern Stacks?”



Lots of “reference” architectures available. They tend not to think about the speed layer since they are focusing on analytics. **Many don’t mention realtime databases… but we can learn from them.**

ANANT

Distributed Realtime Components

DataStax Enterprise

cassandra

DataStax Astra

Cloud Dataproc

Spark

kafka

PULSAR

NATS

SCYLLA.

yugabyteDB

Amazon EMR

Flink

CONFLUENT

presto

play akka

alpakka

Starburst

TiDB

Azure Cosmos DB

amazon Kinesis

pinot

Cloud Spanner

AWS Lambda

databricks

star-tree

ClickHouse

Cockroach DB

ANANT

amazon DynamoDB

To create globally distributed and real time platforms, we need to use distributed realtime technologies to build your platform. Here are some. Which ones should you choose?

Playbook / Framework

Framework Components

- Major Components
  - **Persistent Queues (RAM/BUS)**
  - **Queue Processing & Compute (CPU)**
  - **Persistent Storage (DISK/RAM)**
  - Reporting Engine (Display)
  - **Orchestration Framework (Motherboard)**
  - **Scheduler (Operating System)**
- Strategies
  - Cloud Native on Google
  - Self-Managed Open Source
  - Self-Managed Commercial Source
  - Managed Commercial Source

Customers want options, so we decided to create a Framework that can scale with whatever Infrastructure and Software strategy they want to use.

Category	Streams / Queues	Processors	Storage	Reporting	Orchestration
Open Source	**Apache Kafka** Apache Pulsar Apache Nifi RabbitMQ ActiveMQ	Kafka Connect Kafka Streams Pulsar.io Apache Nifi CLR Threads JVM Threads JS Threads Alpakka/Akka **Spark (Scala, Java)**	Apache Cassandra Apache Solr Elasticsearch MySQL Redis Apache Iceberg Delta Lake	Redash Metabase Zeppelin Jupyter + SQL **SparkSQL** Presto	Terraform Ansible Puppet Chef Docker Kubernetes Airflow
Google Native	**Google** Pub/Sub	**Google** Cloud Tasks Dataflow DataProc	**Google** Filestore Bigtable Spanner	**Google** BigQuery	**Google** Cloudrun Build GKE
Commercial Commercial Open Source Managed Open Source	MuleSoft **Confluent** **Luna Pulsar** **Lenses** **Confluent Cloud** **Aiven Cloud**	**DSE Analytics** **Confluent** MuleSoft Talend **Serverless** **Databricks**	**DSE Cassandra** **DSE Search** **DSE Graph** **Databricks** **Delta Lake** **Datastax Astra**	Tableau Microstrategy + **DSE AlwaysOnSQL** Snowflake Starburst Presto Dremio	Terraform Ansible Puppet Chef **Liquibase** **DSE OpsCenter** **LCM**

ANANT

Playbook for **Modern Open Data Platform**

Platform Design

Discovery (Inventory)

- People
- Process
- Information (Objects)
- Systems (Apps)

Evaluate **Framework**

User Experience

- No-Code/Low Code Apps/Form Builders
- Automatic API Generator/Platform
- Customer App/API Framework

Cloud

- Public
- Private
- Hybrid

Data

- Data:**Object**
- Data:**Stream**
- Data:**Table**
- Data:**Index**
- Processor:**Batch**
- Processor:**Stream**

DevOps

- Infrastructure as Code
- Systems Automation
- Application CICD

DataOps

- ETL/ELT/EtLT
- Reverse ETL
- Orchestration

Execute **Approach**

Architecture (Design)

- Cloud
- Data
- DevOps
- DataOps

Engineering

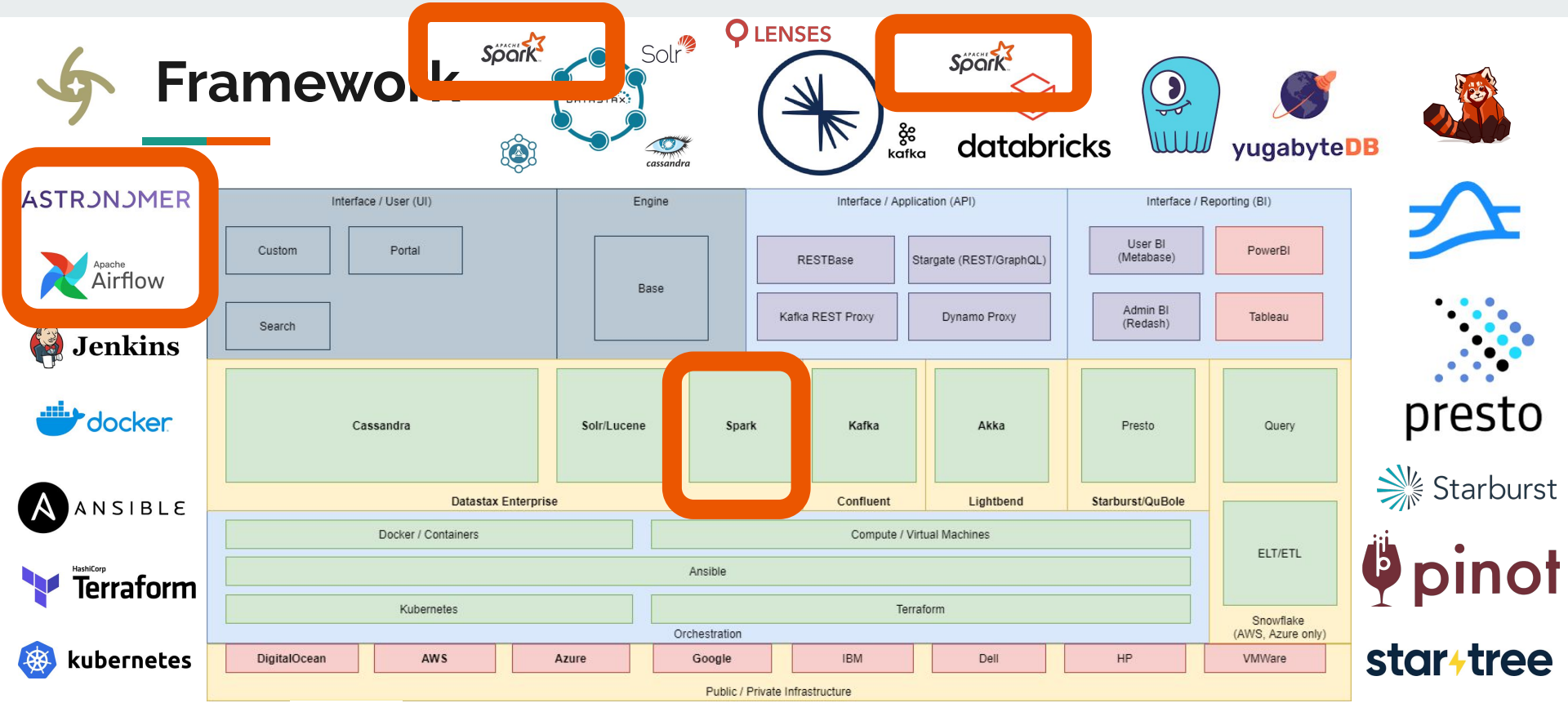
- Configuration
- Scripting
- Programming

Operation

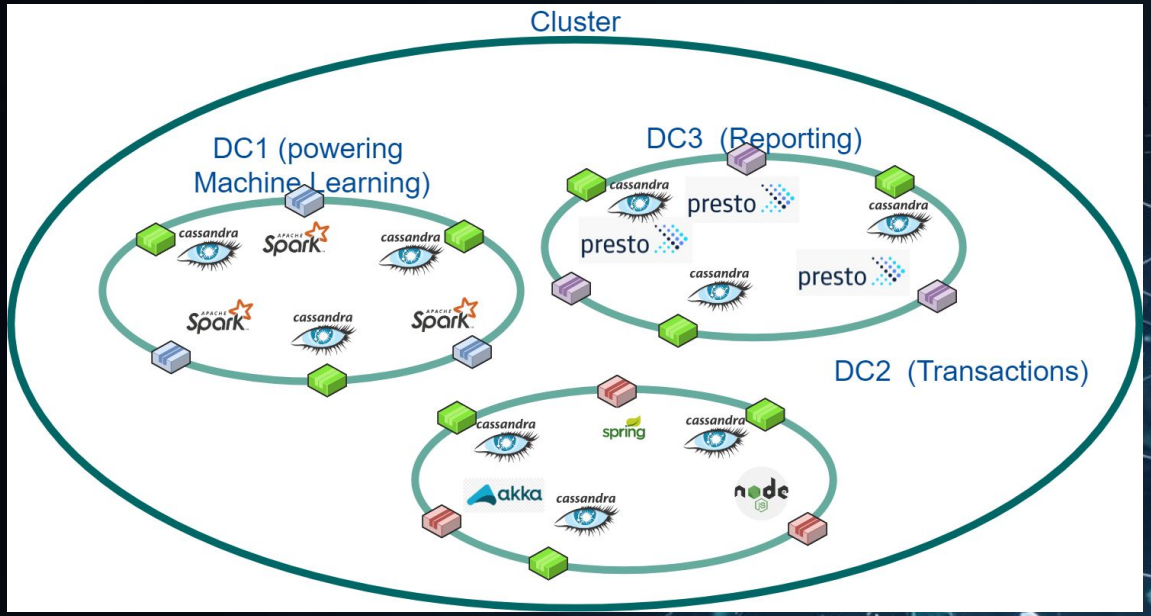
- Setup / Deploy
- Monitoring/Alerts
- Administration

ANANT

Framework


Interface / User (UI)
Custom
Portal
Search
Engine
Base
Interface / Application (API)
RESTBase
Stargate (REST/GraphQL)
Kafka REST Proxy
Dynamo Proxy
Interface / Reporting (BI)
User BI (Metabase)
PowerBI
Admin BI (Redash)
Tableau
Cassandra
Solr/Lucene
Spark
Kafka
Akka
Presto
Query
Datastax Enterprise
Confluent
Lightbend
Starburst/QuBole
Docker / Containers
Compute / Virtual Machines
Ansible
Kubernetes
Terraform
Orchestration
ELT/ETL
Snowflake (AWS, Azure only)
DigitalOcean
AWS
Azure
Google
IBM
Dell
HP
VMWare
Public / Private Infrastructure

Thinking about Cassandra as a Data Fabric



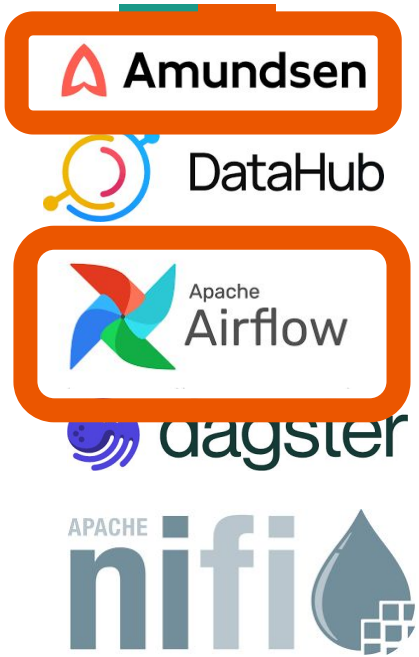
XDCR: Cross datacenter replication is the ultimate data fabric.

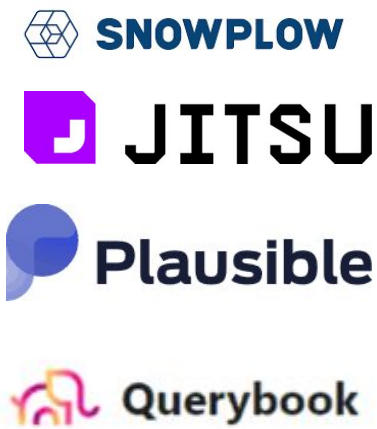
Resilience, performance, availability, and scale.

Made widely available by Cassandra and Couchbase

ANANT

Data Modernization / Automation / Integration





ANANT

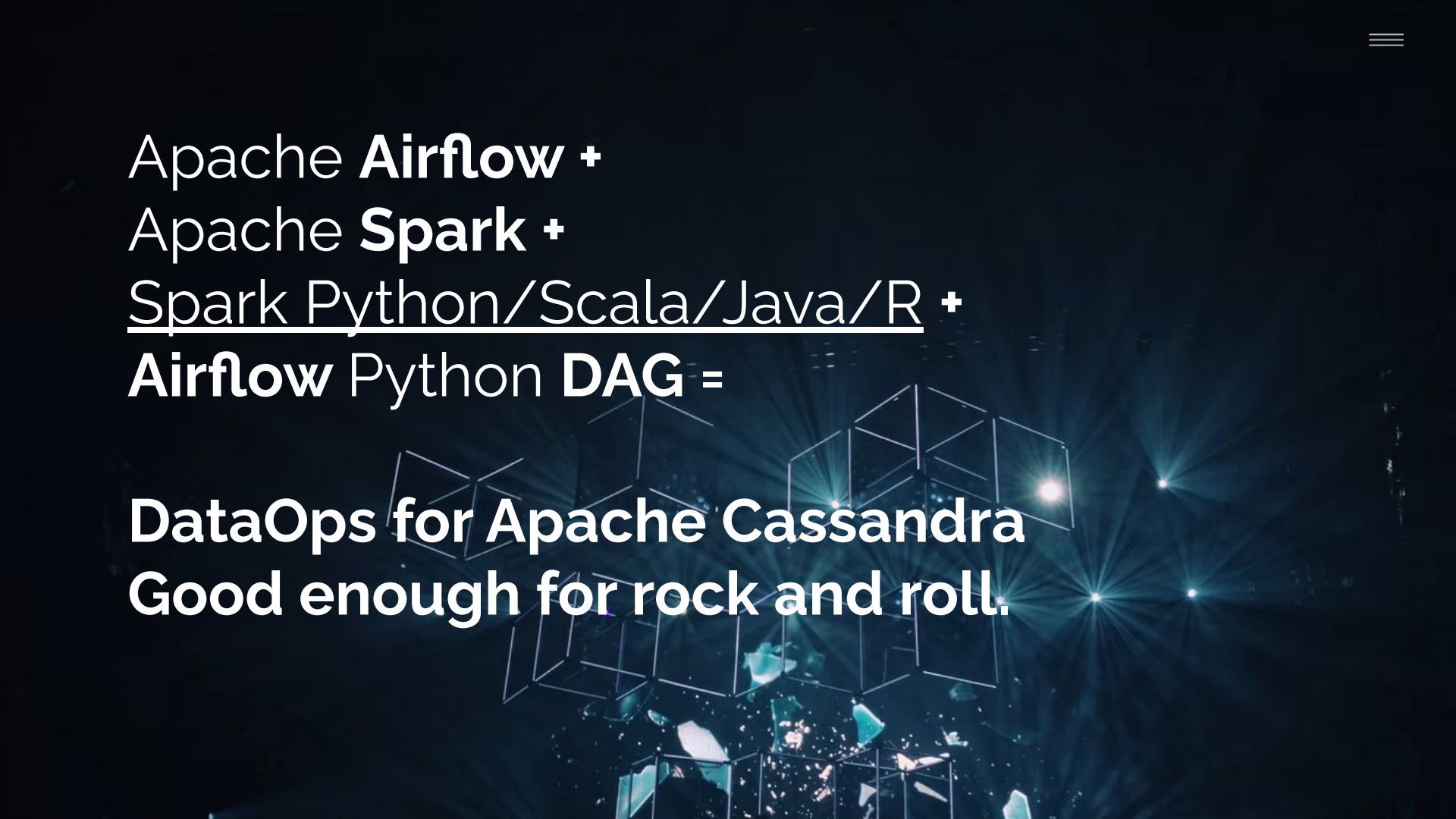
In addition to vastly scalable tools, there are also modern innovations that can help teams automate and maximize human capital by making data platform management easier.

Playbook / Approach

Approach

Setup	Training	Administration	Customization	Knowledge Management
Dev / Ops / Arch Team	Dev / Ops / Arch Team	Dev / Ops / Arch Team	Dev / Ops / Arch Team	Dev / Ops / Arch Team
Understanding current / future needs	Discovery : Understand skill gaps	Service management	Self Service process	Platform Architecture
Mapping a next viable architecture	Curriculum Planning	Service transition	Testing environments	Platform Operations Runbook
Deciding on Infrastructure	Architecture Training	Incident management	Service transition	Platform Implementation Guide
Install & Stress Test	Implementation Training	Data / Disaster Recovery Plan	Service strategy	Issue resolution knowledge
Orchestration Automation	Operations Training	Continuous monitoring	Continuous testing	Public / Internal knowledge sources
Continuous Integration & Delivery	Ongoing mentoring	Continuous testing	Change management	Continuous knowledge creation

ANANT


Apache **Airflow** +
Apache **Spark** +
Spark Python/Scala/Java/R +
Airflow Python **DAG** =
DataOps for Apache Cassandra
Good enough for rock and roll.

Apache Airflow

- A tool for scheduling and automating workflows and tasks
- Good for automating repeated processes
  - Common ETL tasks
  - Machine learning model training
- Write workflows in Python
  - Tools interactable via Python should work as well
  - Define dependencies for different sections of workflows
  - Workflows are DAG of tasks
- Schedule workflows or execute the processes by hand
  - Cron-like syntax or frequency tags
- Monitor tasks and collect/view logs

Apache Spark

- Unified analytics engine for large-scale data processing.
- Achieves high performance for both batch and streaming data, using a state-of-the-art DAG scheduler, a query optimizer, and a physical execution engine.
- Offers over 80 high-level operators that make it easy to build parallel apps. And you can use it interactively from the Scala, Python, R, and SQL shells.
- Powers a stack of libraries including SQL and DataFrames, MLlib for machine learning, GraphX, and Spark Streaming. You can combine these libraries seamlessly in the same application.
- You can run Spark using its standalone cluster mode, on EC2, on Hadoop YARN, on Mesos, or on Kubernetes. Access data in HDFS, Alluxio, Apache Cassandra, Apache HBase, Apache Hive, and hundreds of other data sources.

Big Data Options

Coldish

- S3
- HDFS
- ADLS
- GFS

Warm

- Hive / *
- Data Warehouse
- Data Lakehouse

Hot

- **Cassandra***
- **Datastax***
- **Scylla***
- **Yugabyte***
- Mongo
- REDIS
- ...

Hot*

- **Astra***
- **Scylla Cloud***
- **YugaByte Cloud***
- **Azure CosmosDB***
- **AWS Keyspaces***
- AWS Dynamo
- Google BigTable
- ...

*** PSSST. These all use CQL!!!**

ANANT

Cleaning Big Data: Same $h1t Different Day

Data Cleaning as part of Data Engineering

- Step 1: Remove duplicate or irrelevant observations
- Step 2: Fix structural errors
- Step 3: Filter unwanted outliers
- Step 4: Handle missing data
- Step 5: Validate and QA

Data Cleaning after the Fact

- Enforce a custom data retention policy (TTL)
- Enforce GDPR / Right to be Forgotten
- Move application, customer, user from one system to another
- Remove x “versions” or “revisions” of data
- Remove test data from a stress test

https://www.tableau.com/learn/articles/what-is-data-cleaning

Cleaning Big Data: In SQL ….

```
9
10  DELETE FROM [dbo].[SalesData]
11  WHERE CustomerId IN
12      (SELECT TOP 10 CustomerId
13       FROM [dbo].[SalesData]
14       ORDER BY OrderDate ASC);
15  GO
16
17
```

100 %

Messages

(13 rows affected)

Data Cleaning in SQL

- Find what do you want to delete.
- Delete it.

ANANT

Cleaning Big Data: In Spark SQL ….

```
SQL

> DELETE FROM events WHERE date < '2017-01-01'

> DELETE FROM all_events
   WHERE session_time < (SELECT min(session_time) FROM good_events)

> DELETE FROM orders AS t1
   WHERE EXISTS (SELECT oid FROM returned_orders WHERE t1.oid = oid)

> DELETE FROM events
   WHERE category NOT IN (SELECT category FROM events2 WHERE date > '2001-01-01')
```

Data Cleaning in SPARK SQL

- What do you want to delete. Delete it.

WARN: Doesn't work with all data in Spark SQL. Only if connector supports **Table Delete**

Deletes the rows that match a predicate. When no predicate is provided, deletes all rows.

This statement is only supported for Delta Lake tables.

https://docs.databricks.com/spark/latest/spark-sql/language-manual/delta-delete-from.html
https://spark.apache.org/docs/latest/api/java/org/apache/spark/sql/connector/catalog/SupportsDelete.html

Cleaning Big Data: Cleaning data in Spark / SQL

Data Cleaning in Spark for Cassandra

- What do you want to delete.
- Delete it.

```
def deleteCategory(keyspace: String, table: String, filter: String): Long = {
  import com.datastax.spark.connector.cql.CassandraConnector
  val dseConnector = CassandraConnector(sc)
  val documentDF = spark
    .read
    .format("org.apache.spark.sql.cassandra")
    .options(Map( "table" -> table, "keyspace" -> keyspace))
    .load
  documentDF.createOrReplaceTempView("documents")
  val deleteDocDF = spark.sql("SELECT * FROM documents "+filter)
  val deletedCount = deleteDocDF.count()
  deleteDocDF.foreachPartition(partition => {
    dseConnector.withSessionDo(session =>
      partition.foreach(log =>
          val delete = s"DELETE FROM "+keyspace+"."+table+" where <primary key>="+log.getString(0) +";"
          session.execute(delete)
      ))
  })
  deletedCount
}
```

https://stackoverflow.com/questions/28563809/delete-from-cassandra-table-in-spark

Cleaning Big Data: Deduping in Spark SQL

Deduping Data in Spark for Cassandra

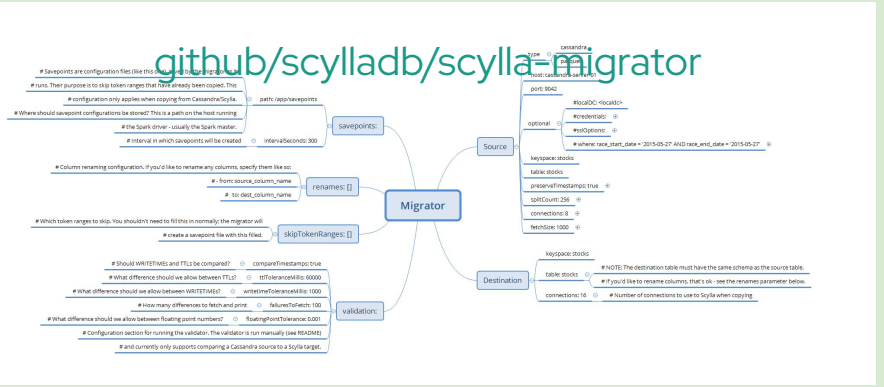
- What do you want to dedupe.
- Do some deduping.
- What you want to delete.
- Delete it.

```
def dedupeTable(keyspace: String, table: String, columns: Array[String], filter: String): Long = {
  import com.datastax.spark.connector.cql.CassandraConnector
  val dseConnector = CassandraConnector(sc)
  val documentDF = spark
    .read
    .format("org.apache.spark.sql.cassandra")
    .options(Map( "table" -> table, "keyspace" -> keyspace))
    .load
//TODO this category filter can be done using SPARK SQL, and you should be able to just pass it a SQL "where clause"
  documentDF.createOrReplaceTempView("documents")
  val docDF = spark.sql("SELECT * FROM documents "+filter)
  val documentDF_temp = docDF.withColumn("comparable", concat_ws(",", (columns.map(c => col(c)): _*)))
  val countsDF = documentDF_temp.rdd.map(rec => (rec(9),1)).reduceByKey(_+_).map(rec => (rec._1.toString, rec._2)).toDF("Compared", "Count")
  val joinedDF = documentDF_temp.join(countsDF, documentDF_temp("comparable")===countsDF("Compared"),
    "left_outer").select(documentDF_temp("doc_id"), documentDF_temp("comparable"), countsDF("Count"))
  val result = joinedDF.where("Count > 1")
  val originals = result.groupBy("comparable").agg(first("doc_id").as("doc_id"), min("count").as("count"))
  val to_Delete = (result.select("doc_id")).except(originals.select("doc_id"))
  val deletedCount = result.count()
  to_Delete.foreachPartition(partition => {
    dseConnector.withSessionDo(session =>
      partition.foreach{ log =>
          val delete = s"DELETE FROM "+keyspace+"."+table+" where doc_id="+log.getString(0) +";"
          session.execute(delete)
      })
  })
  deletedCount
}
```

ANANT

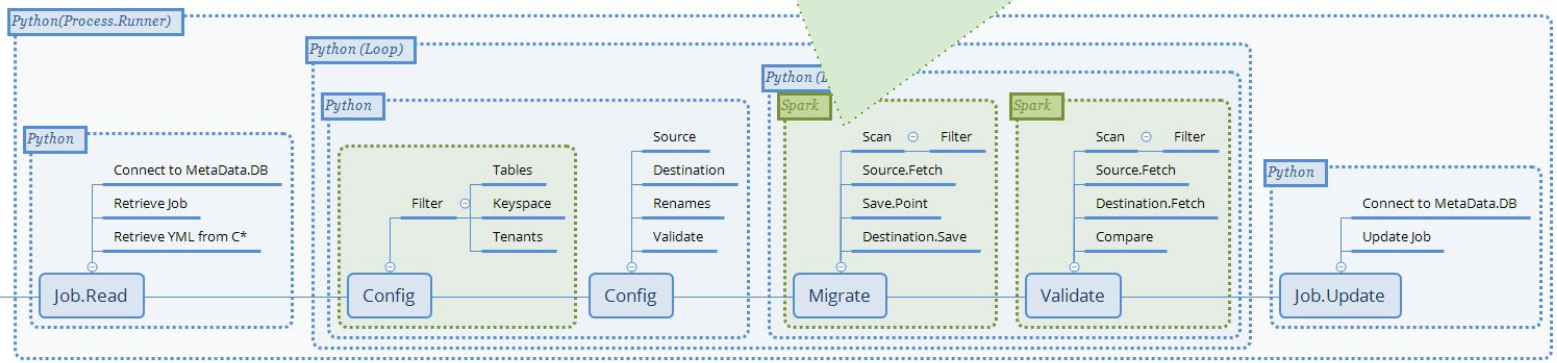
Airflow DAG to Migrate Cassandra Data

github/scylladb/scylla-migrator



Apache Airflow

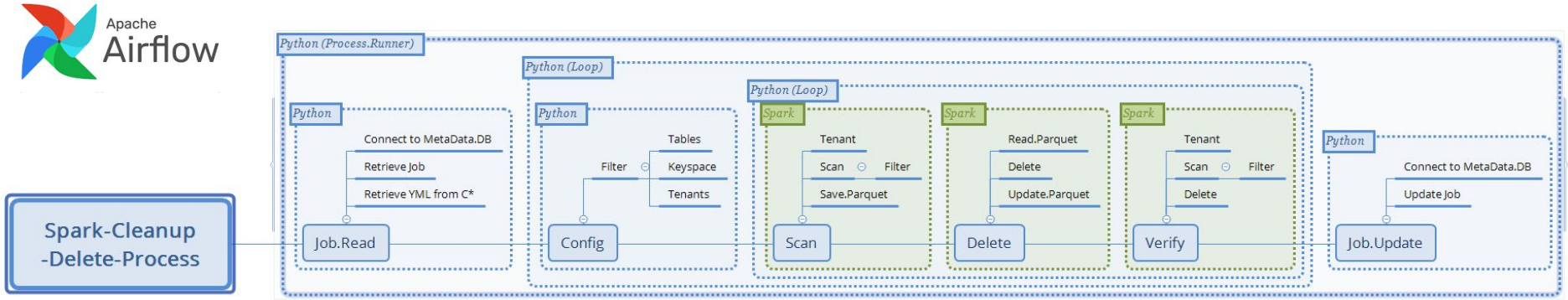
Spark-Migrator -Process



Airflow can help us take any data process, compiled, interpreted etc, coordinate the steps as a DAG (“Directed Acyclic Graph”) and then to make it even more awesome, parametrize it either via the Airflow GUI or our own table somewhere.

ANANT

Airflow DAG to Clean Cassandra Data



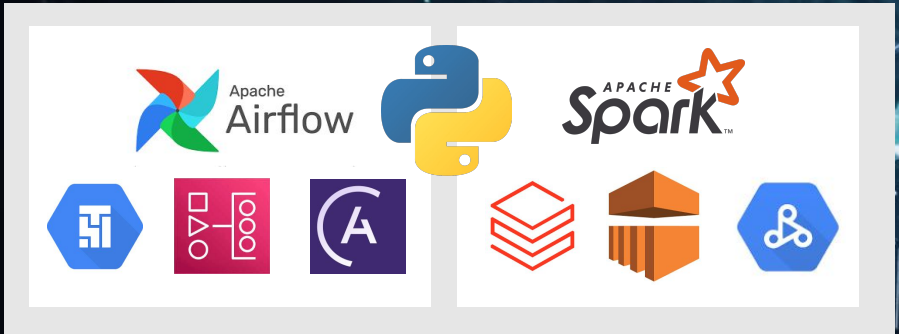
Since we write abstracted code, we can replace the “Migrator” process with a Delete, Dedupe, Validate. Whatever.
Airflow allows us to reuse conventions we set in a team for large scale operations, and most importantly... make it easy for people to run Data Operations like this without being Cassandra , Spark, Python experts.

ANANT

Considerations for Spark/Airflow Solution

Considerations for Spark

- Figure out if you want to **manage it / run it.**
- Not all Spark code is created equally.
- Not all Spark languages run the same.
- Compiled Jobs with input parameters can work better in the long run, less room for code drift.
- Don't let people do adhoc delete operations until and unless it's absolutely necessary.
- Who has access to it?



Considerations for Airflow

- Figure out if you are going to manage it / run it.
- Figure out for whom you are going to run it (Platform, Environment, Stack, App, Customer?)
- Not all DAGs just work. Sometimes they need tweaking across Environments, Stacks, Apps, Customers.
- The same DAG may fail over time. Need to watch execution times.
- Who has access to it?

ANANT

Key Takeaways for Cassandra Data Operations

Don’t reinvent the wheel.

Use Apache Spark

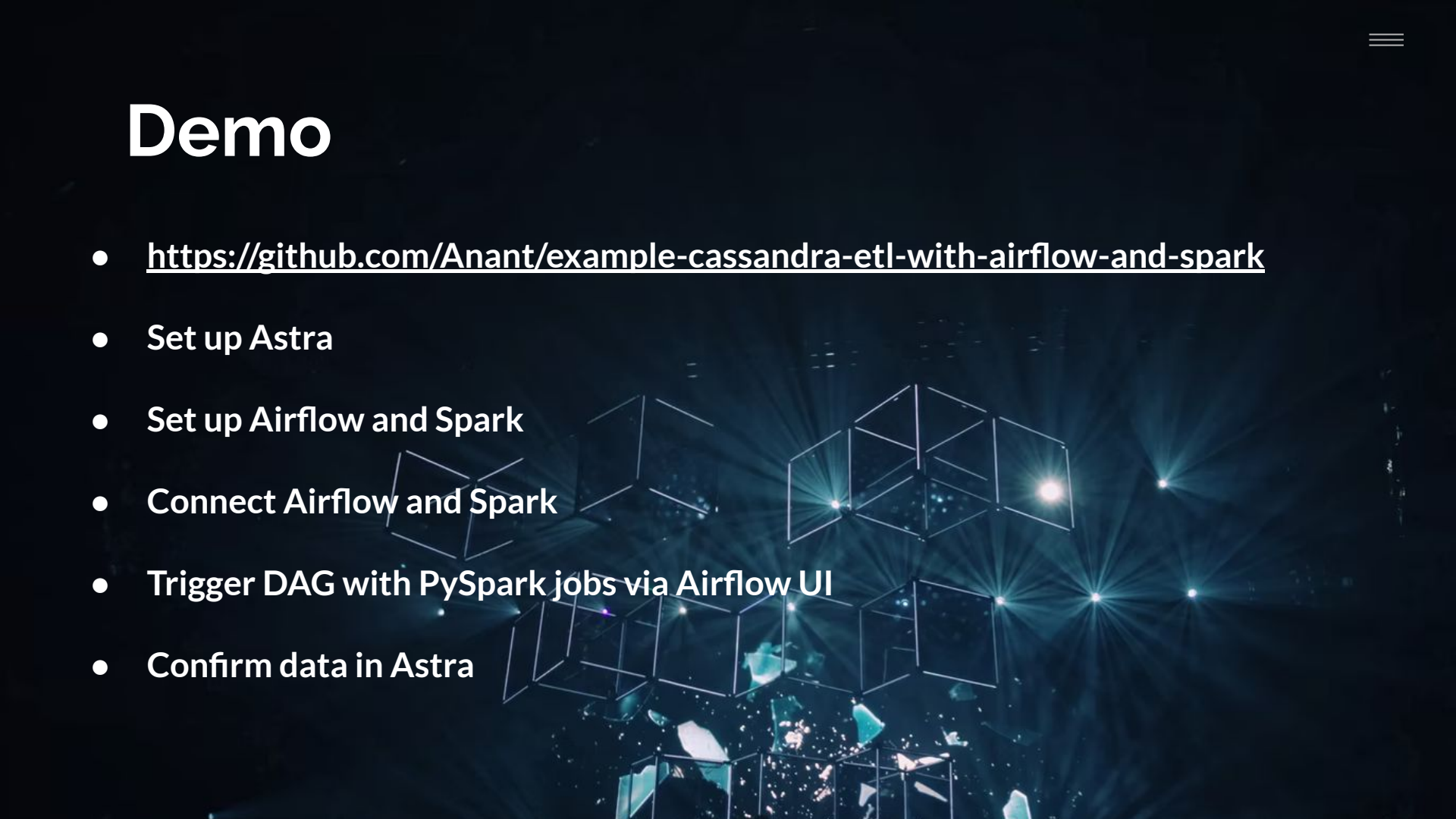
Use a Scheduler (Apache Airflow w/ Python)

- You can look, but Apache Spark is basically it. Look no further.
- Learn Spark, Python / Scala is fine. Just start using Apache Spark.
- Airflow, Jenkins, Luigi, Prefect, any scheduler can work, but Airflow has been proven for this.
- Airflow works with more than just Apache Cassandra, Apache Spark.. There are numerous Connections and Operators.

ANANT

Demo

- https://github.com/Anant/example-cassandra-etl-with-airflow-and-spark
- Set up Astra
- Set up Airflow and Spark
- Connect Airflow and Spark
- Trigger DAG with PySpark jobs via Airflow UI
- Confirm data in Astra



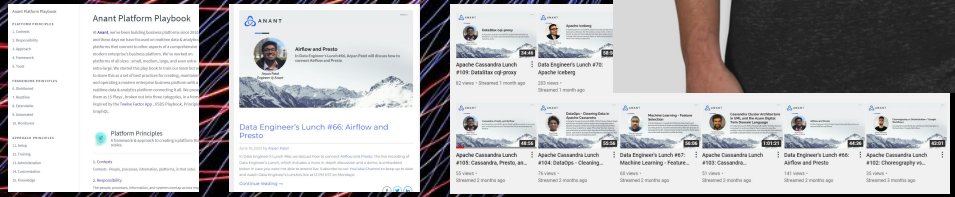
Thank you and Dream Big.

Hire us

- **Design** Workshops
- **Innovation** Sprints
- **Service Catalog**

Anant.us

- Read our Playbook
- Join our Mailing List
- Read up on Data Platforms
- Watch our Videos
- Download Examples



www.anant.us | solutions@anant.us | (855) 262-6826
3 Washington Circle, NW | #301 | Washington, DC 20037

ANANT