Running Apache Airflow Workflows as ETL Processes on Hadoop

By: Robert Sanders

CLAIRVOYANT

Agenda

- What is Apache Airflow?
  - Features
  - Architecture
  - Terminology
  - Operator Types
- ETL Best Practices
  - How they’re supported in Apache Airflow
- Executing Airflow Workflows on Hadoop
- Use Cases
- Q&A

Robert Sanders

- Big Data Manager, Engineer, Architect, etc.
- Work for Clairvoyant LLC
- 5+ Years of Big Data Experience
- Email: robert.sanders@clairvoyantsoft.com
- LinkedIn: https://www.linkedin.com/in/robert-sanders-61446732
- Slide Share: http://www.slideshare.net/RobertSanders49

What’s the problem?

- As a Big Data Engineer you work to create jobs that will perform various operations
  - Ingest data from external data sources
  - Transformation of Data
  - Run Predictions
  - Export data
  - Etc.
- You need to have some mechanism to schedule and run these jobs
  - Cron
  - Oozie
- Existing Scheduling Services have a number of limitations that make them difficult to work with and not usable in all instances

What is Apache Airflow?

- **Airflow is an Open Source platform to programmatically author, schedule and monitor workflows**
    - Workflows as Code
    - Schedules Jobs through Cron Expressions
    - Provides monitoring tools like alerts and a web interface
- Written in Python
    - As well as user defined Workflows and Plugins
- Was started in the fall of 2014 by Maxime Beauchemin at Airbnb
- Apache Incubator Project
    - Joined Apache Foundation in early 2016
    - https://github.com/apache/incubator-airflow/

Why use Apache Airflow?

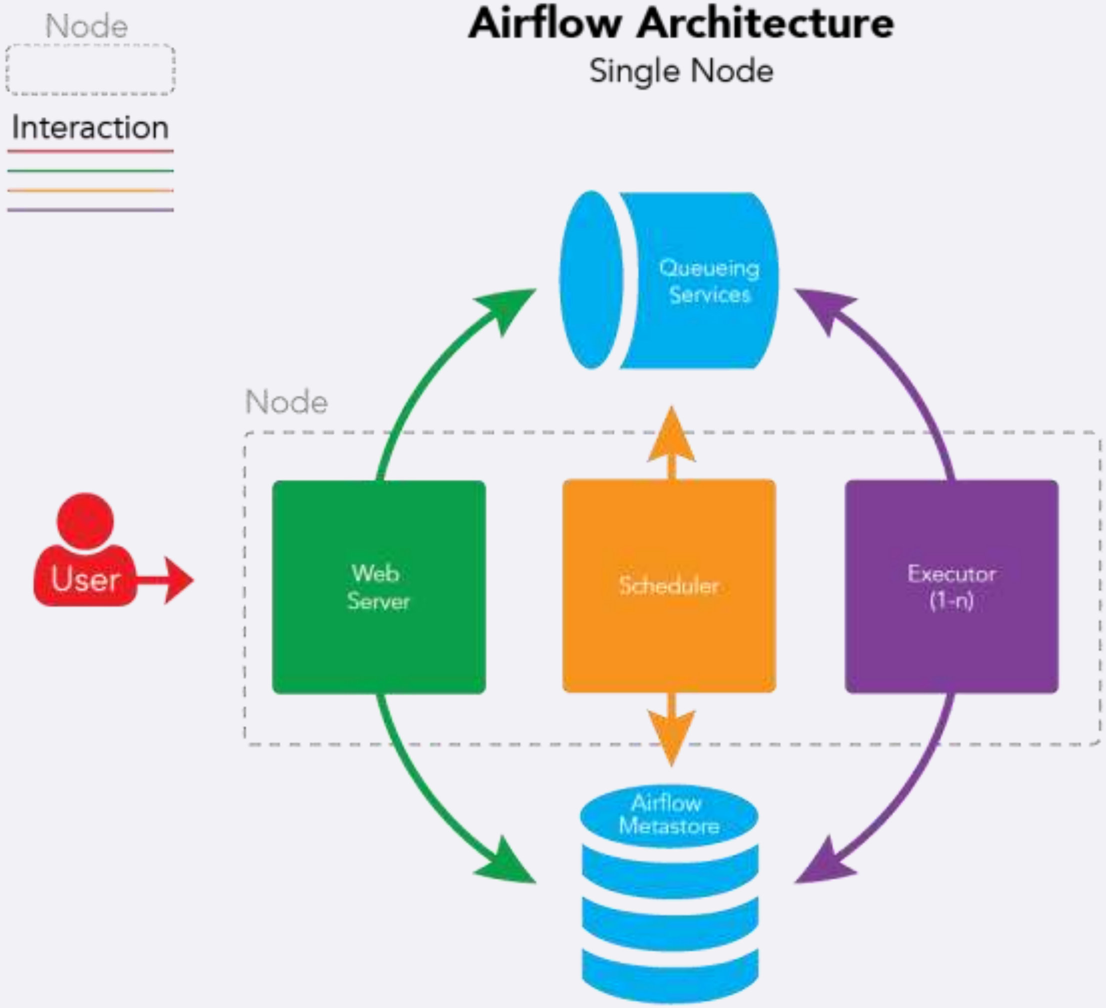
- Define Workflows as Code
  - Makes workflows more maintainable, versionable, and testable
  - More flexible execution and workflow generation
- Lots of Features
- Feature Rich Web Interface
- Worker Processes Scale Horizontally and Vertically
  - Can be a cluster or single node setup
- Lightweight Workflow Platform

Apache Airflow Features (Some of them)

- Automatic Retries
- SLA monitoring/alerting
- Complex dependency rules: branching, joining, sub-workflows
- Defining ownership and versioning
- Resource Pools: limit concurrency + prioritization
- Plugins
  - Operators
  - Executors
  - New Views
- Built-in integration with other services
- Many more…

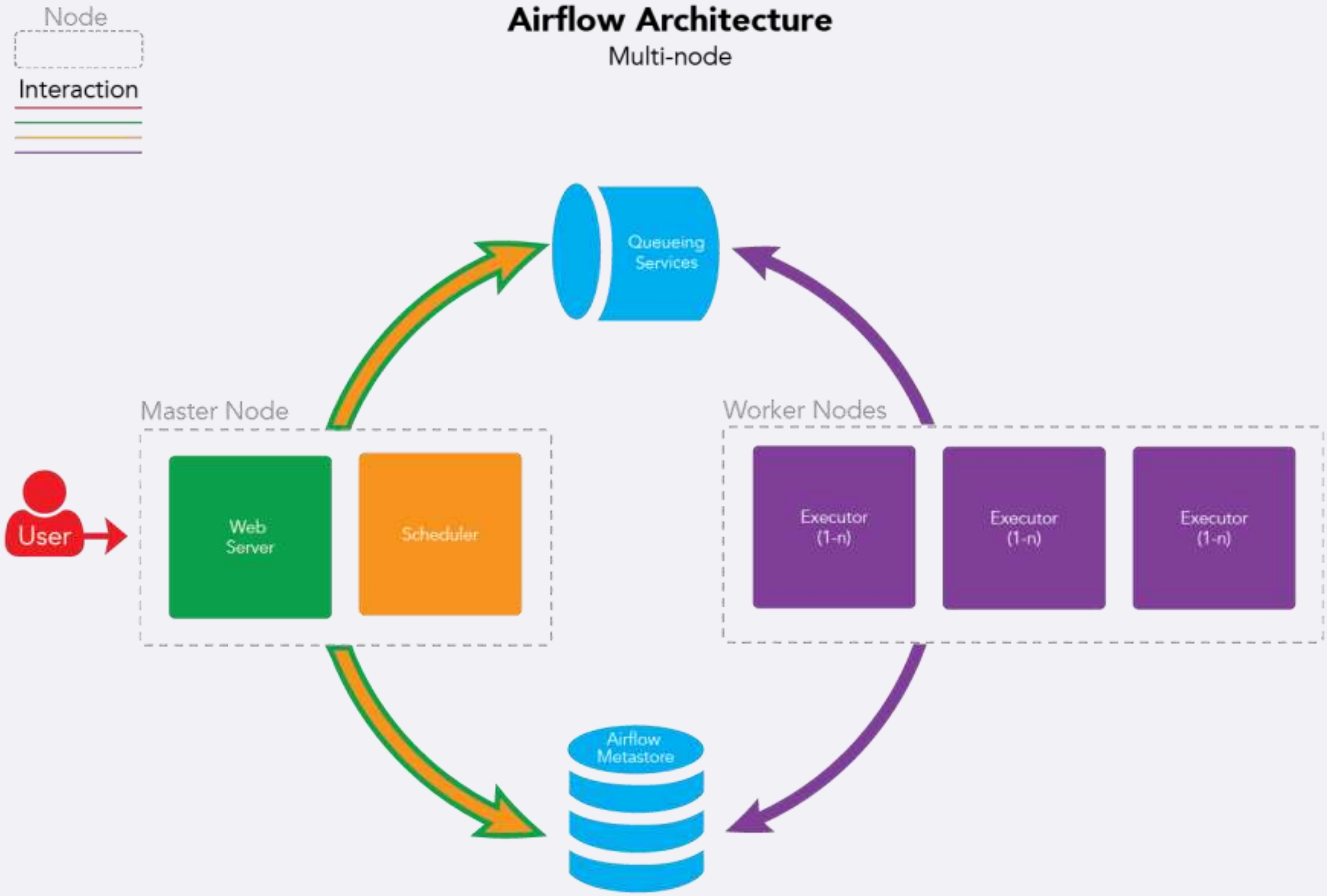
Airflow Architecture

Single Node



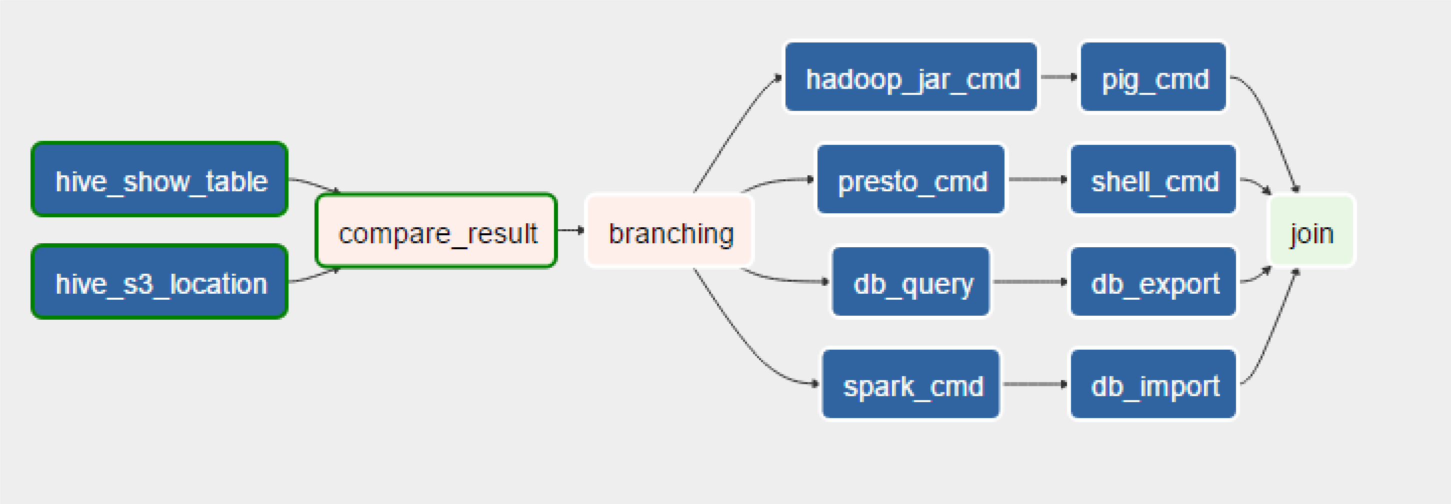
Airflow Architecture

Multi-node



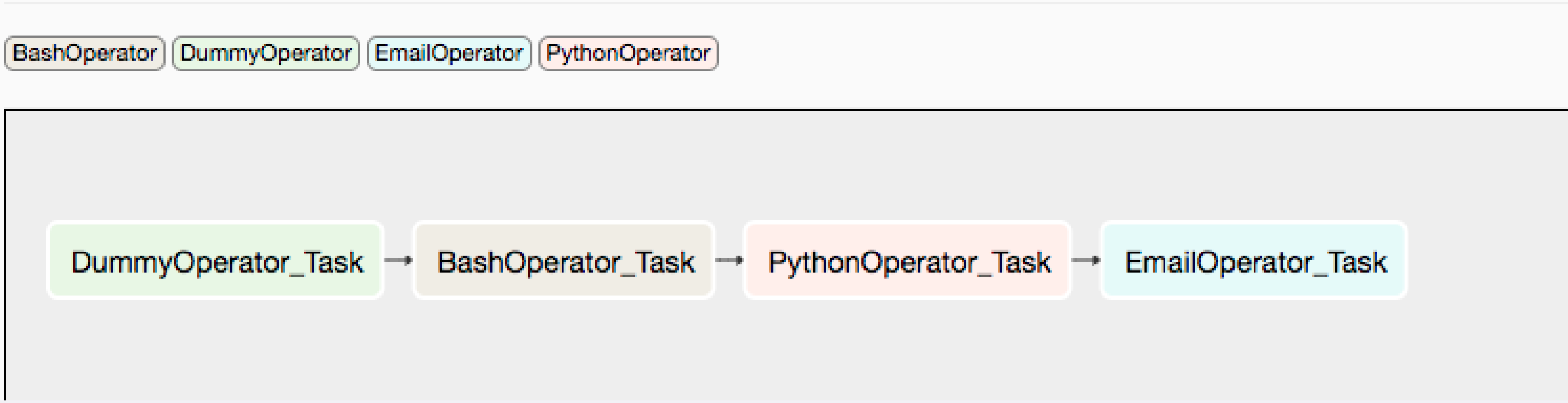
What is a DAG?

- **D**irected **A**cyclic **G**raph
  - A finite directed graph that doesn't have any cycles
- A collection of tasks to run, organized in a way that reflects their relationships and dependencies
  - Defines your Workflow



What is an Operator?

- An operator describes a single task in a workflow
- Operators allow for generation of certain types of tasks that become nodes in the DAG when instantiated
- All operators derive from BaseOperator and inherit many attributes and methods that way



Workflow Operators (Sensors)

- A type of operator that keeps running until a certain criteria is met
- Periodically pokes
- Parameterized poke interval and timeout
- Example
  - HdfsSensor
  - HivePartitionSensor
  - NamedHivePartitionSensor
  - S3KeyPartition
  - WebHdfsSensor
  - Many More…

Workflow Operators (Transfer)

- Operator that moves data from one system to another
- Data will be pulled from the source system, staged on the machine where the executor is running and then transferred to the target system
- Example:
  - HiveToMySqlTransfer
  - MySqlToHiveTransfer
  - HiveToMsSqlTransfer
  - MsSqlToHiveTransfer
  - S3ToHiveTransfer
  - Many More…

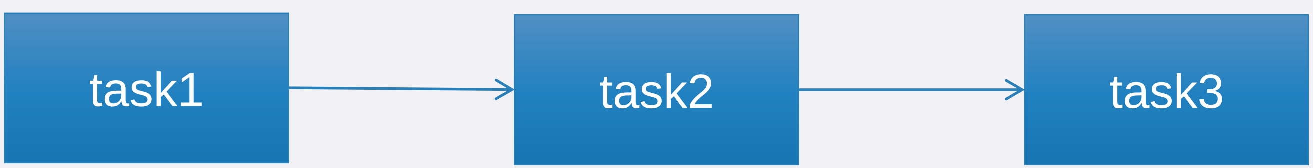
Defining a DAG

```
from airflow.models import DAG
from airflow.operators import …
from datetime import datetime, timedelta

default_args = dict(
      'owner'='Airflow’,
      'retries': 1,
      'retry_delay': timedelta(minutes=5),
)
# Define the DAG
dag = DAG('dag_id', default_args=default_args, schedule_interval='0 0 * * *')

# Define the Tasks
task1 = BashOperator(task_id='task1', bash_command="echo 'Task 1'", dag=dag)
task2 = BashOperator(task_id='task2', bash_command="echo 'Task 2'", dag=dag)
task3 = BashOperator(task_id='task3', bash_command="echo 'Task 3'", dag=dag)

# Define the task relationships
task1.set_downstream(task2)
task2.set_downstream(task3)
```



Defining a DAG (Dynamically)

```
dag = DAG('dag_id', default_args=default_args, schedule_interval='0 0 * * *')

last_task = None
for i in range(1, 3):
    task = BashOperator(
        task_id='task' + str(i),
        bash_command="echo 'Task" + str(i) + "'",
        dag=dag)
    if last_task is None:
        last_task = task
    else:
        last_task.set_downstream(task)
        last_task = task
```

ETL Best Practices (Some of Them)

- Load Data Incrementally
  - Operators will receive an execution_date entry which you can use to pull in data since that date
- Process historic Data
  - Backfill operations are supported
- Enforce Idempotency (retry safe)
- Execute Conditionally
  - Branching, Joining
- Understand SLA’s and Alerts
  - Alert if Failures
- Sense when to start a task
  - Sensor Operators
- Build Validation into your Workflows

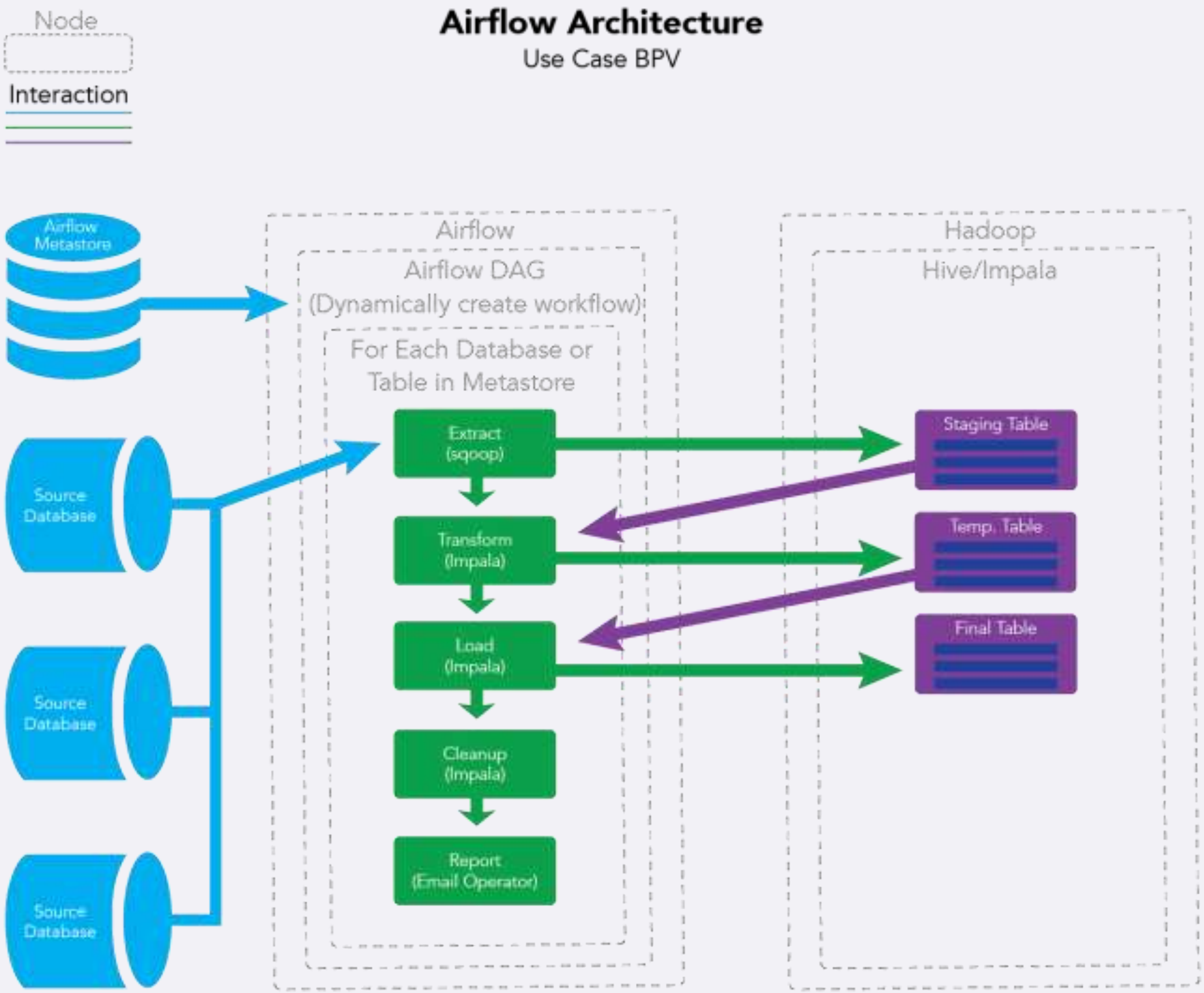
Executing Airflow Workflows on Hadoop

- Airflow Workers should be installed on a edge/gateway nodes
  - Allows Airflow to interact with Hadoop related commands
  - Utilize the BashOperator to run command line functions and interact with Hadoop services
- Put all necessary scripts and Jars in HDFS and pull the files down from HDFS during the execution of the script
  - Avoids requiring you to keep copies of the scripts on every machine where the executors are running
- Support for Kerborized Clusters
  - Airflow can renew Kerberos tickets for itself and store it in the ticket cache

Use Case (BPV)

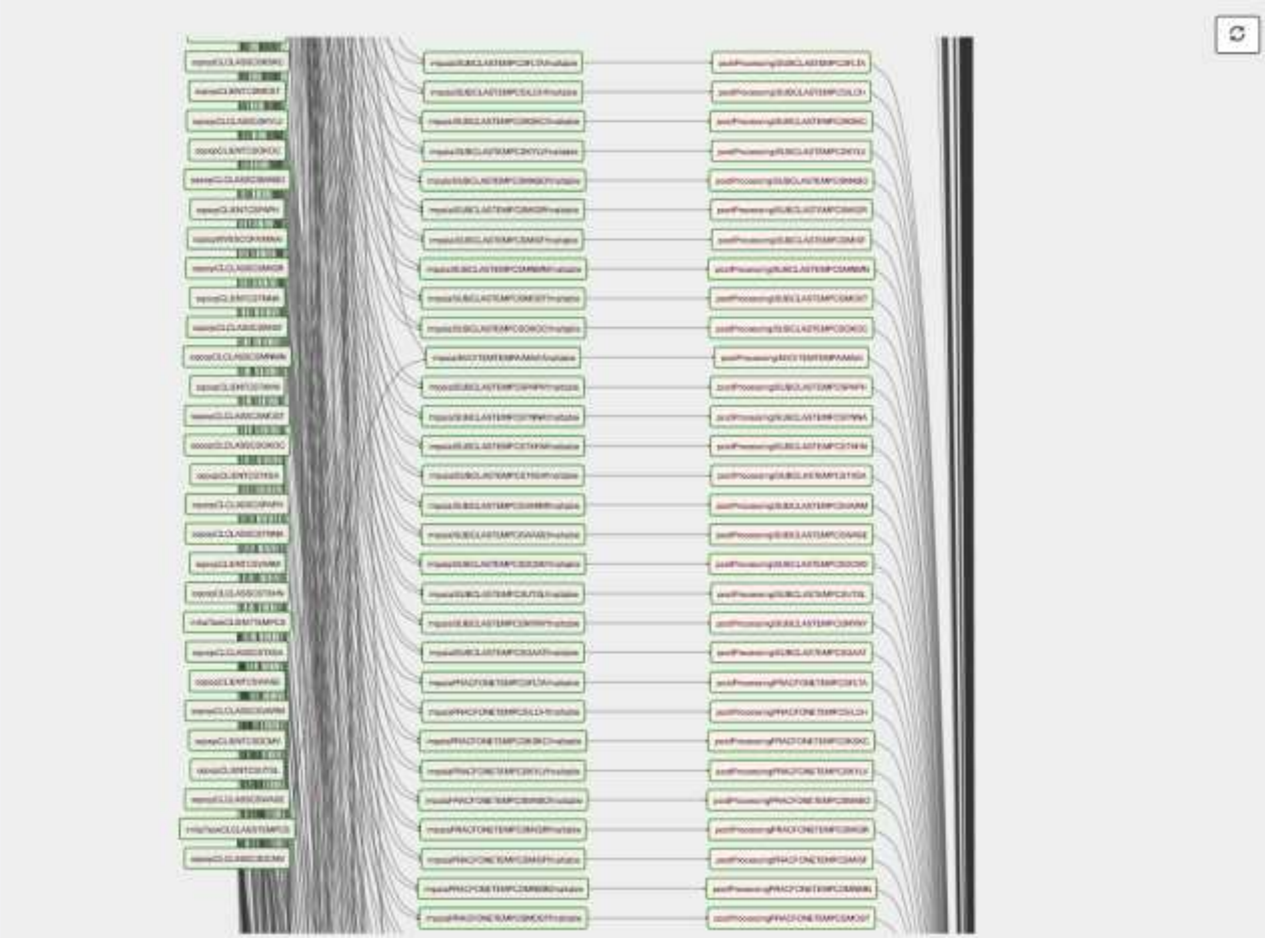
- Daily ETL Batch Process to Ingest data into Hadoop
  - Extract
    - 23 databases total
    - 1226 tables total
  - Transform
    - Impala scripts to join and transform data
  - Load
    - Impala scripts to load data into common final tables
- Other requirements
  - Make it extensible to allow the client to import more databases and tables in the future
  - Status emails to be sent out after daily job to report on success and failures
- Solution
  - Create a DAG that dynamically generates the workflow based off data in a Metastore

Use Case (BPV) (Architecture)



Use Case (BPV) (DAG)

100 foot view

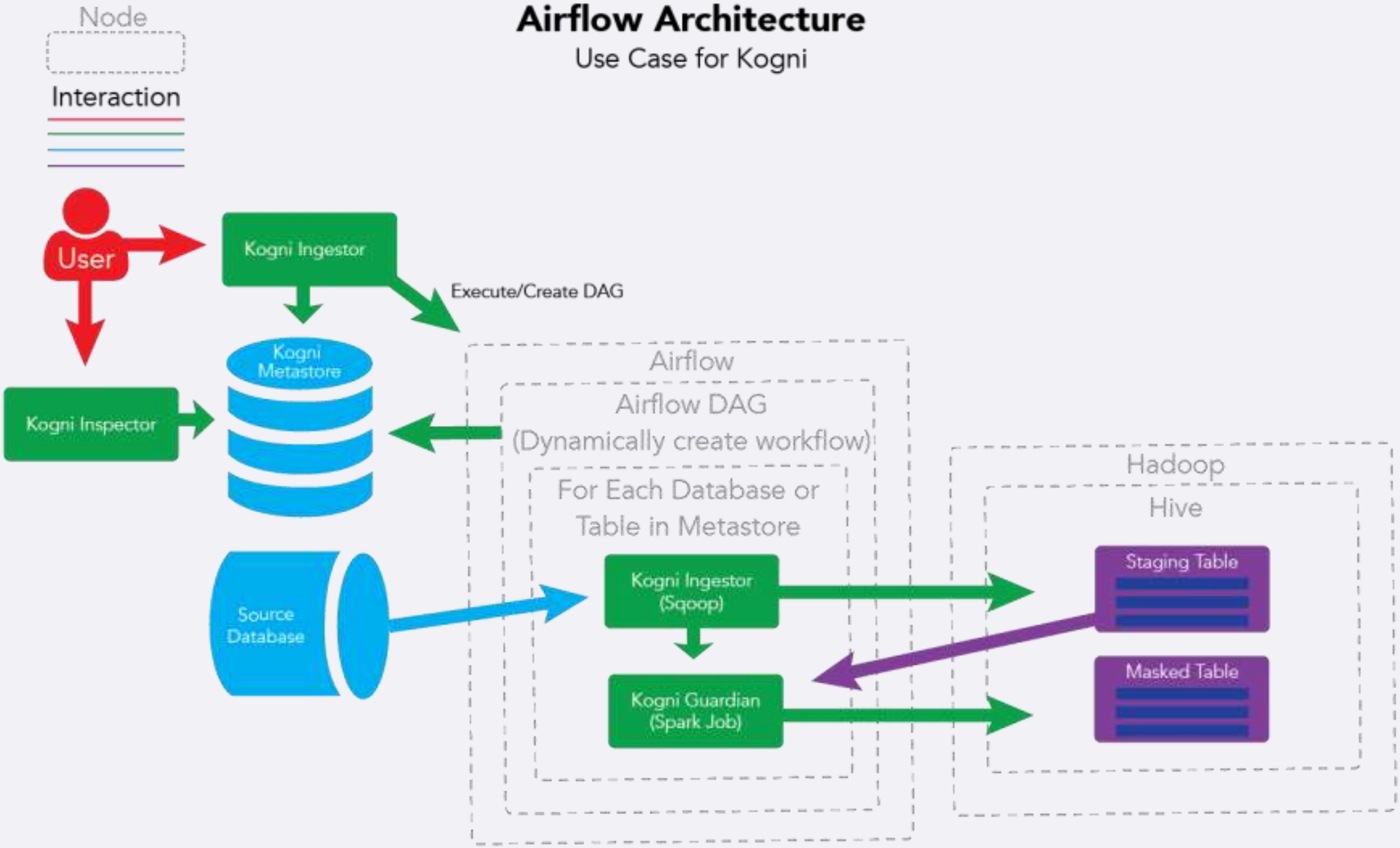


10,000 foot view

Use Case (Kogni)

- New Product being built by Clairvoyant to facilitate:
  - kogni-inspector – Sensitive Data Analyzer
  - kogni-ingestor – Ingests Data
  - kogni-guardian – Sensitive Data Masking (Encrypt and Tokenize)
  - Others components coming soon
- Utilizes Airflow for Data Ingestion and Masking
- Dynamically creates a workflow based off what is in the Metastore
- Learn More: http://kogni.io/

Use Case (Kogni) (Architecture)



References

- https://pythonhosted.org/airflow/
- https://gtoonstra.github.io/etl-with-airflow/principles.html
- https://github.com/apache/incubator-airflow
- https://media.readthedocs.org/pdf/airflow/latest/airflow.pdf

Q&A