Sunbeam Infotech

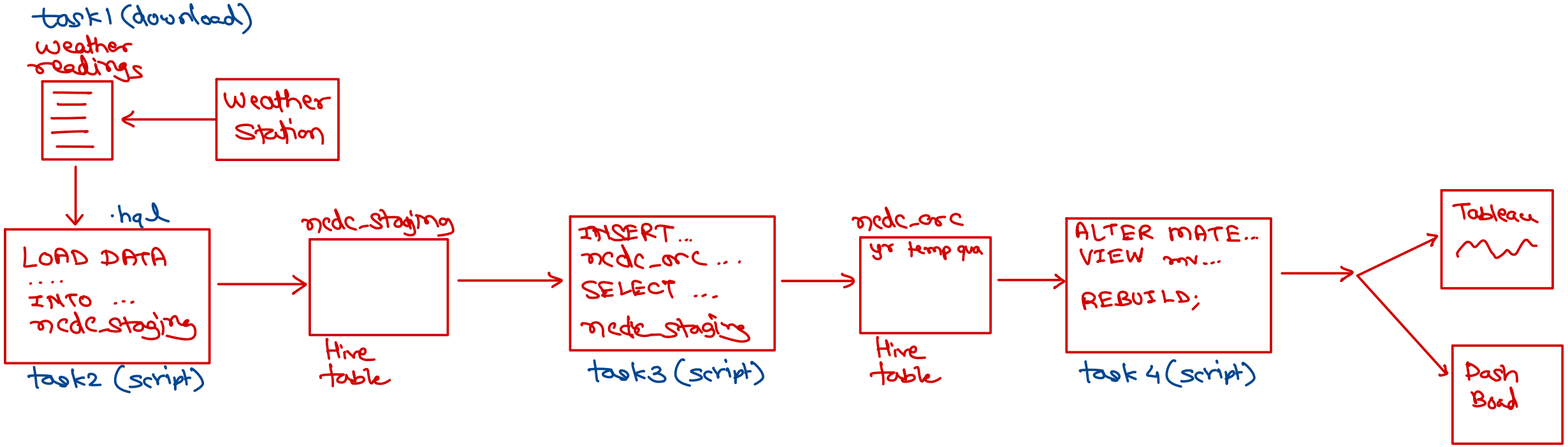
Introduction

Free & open source

- A platform created by community to programmatically author, schedule and monitor workflow. → set of tasks to be done is a specific sequence.
- Airflow is used to develop, orchestrate and monitor complex ETL pipelines. → Python → horizontal scaling → web ui (port 8080)
- Airflow is completely developed in Python, so it can work with any of the Python library. → kafka-python, sklearn, ..
- Initiated by Airbnb in 2014 and open-sourced under Apache in 2016. → github - 29K stars
- Apache top level project in 2019.
- Used by more than 250 companies world-wide - Amazon, Citi, JPMorgan, Salesforce, Drillinginfo, Unitedhealth, ...
- Managed services in GCP and AWS cloud.

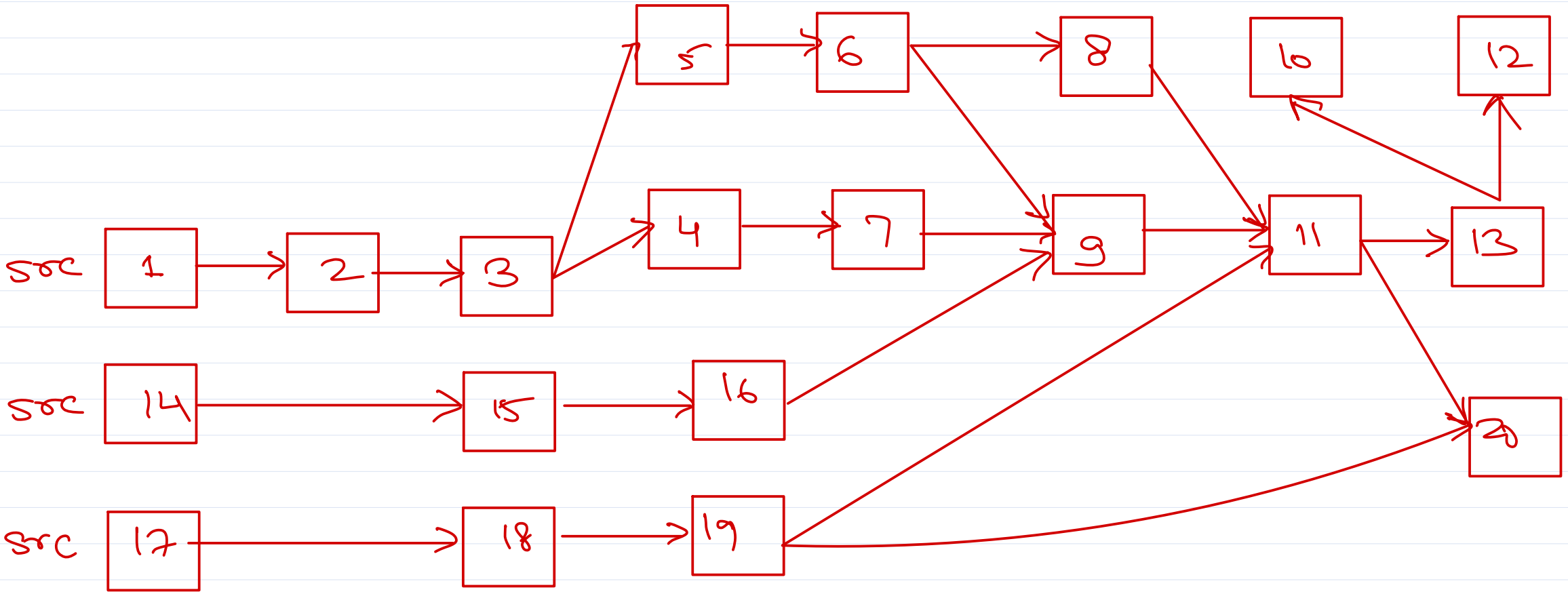
AWS managed services

1. EMR
2. Redshift
3. EKS
4. Glue
5. S3
6. RDS
7. Cloud watch
8. Kinesis
9. Beanstalk
10. Airflow


task1 (download)
weather readings
Weather Station
.hql
LOAD DATA
....
INTO ...
ncdc_staging
task2 (script)
ncdc_staging
Hive table
INSERT...
ncdc_orc...
SELECT ...
ncdc_staging
task3 (script)
ncdc_orc
yr temp qua
Hive table
ALTER MATE...
VIEW mv...
REBUILD;
task 4 (script)
Tableau
Dash Boad

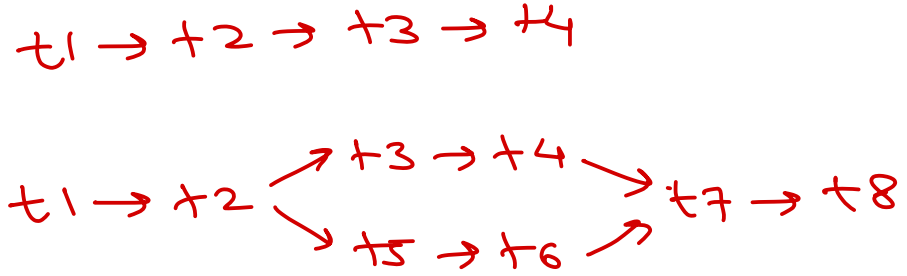
Traditional ETL

- Extract, Transform and Load
- Common schedulers: cron, oozie (hadoop), luigi (spotify)
- BigData has complex pipelines.
- Cron limitations
  - Error handling
  - Not maintainable (runs on a machine) no scaling
  - Execution dependency (delayed tasks)
  - Transperency (No centralized log)
  - Task tracking (No centralized monitoring)
  - Handle historical data
- Oozie
  - Error handling (n retries)
  - Execution dependency
  - Better tracking & monitoring
  - XML based jobs - difficult to read/understand

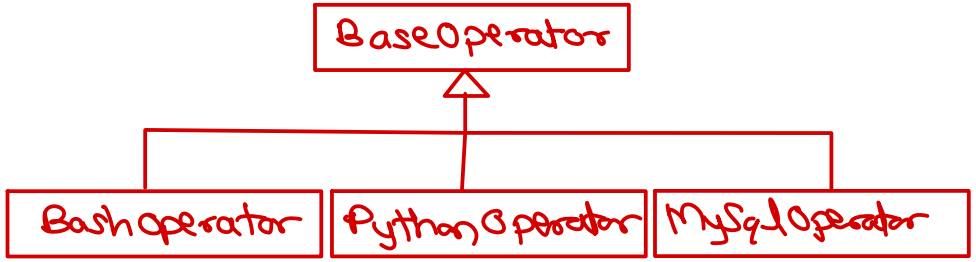

src
1
2
3
5
6
8
10
12
4
7
9
11
13
src
14
15
16
20
src
17
18
19

Airflow Terminologies

- DAG
  - Directed Acyclic Graph
  - Vertices are tasks and Edges are dependencies
  - Can be parallelized.



- Operators
  - Single dedicated task in workflow/DAG.
  - Example: Bash command, Python Function, Database Operation, Email send, etc.



- Task
  - Instantiated operator and assigned to some worker.
  - Failed task can be retried.
- Workflow
  - Sequence of task arranged in control dependencies.
  - Workflow and DAG words are interchangeable.

business terminology

programming terminology

Airflow Workflow

- Airflow workflows are written as Python code in form of DAG.
  - DAG includes multiple tasks in form of operators.
  - Operators are connected in complex sequence.
- DAG implemented in Python, but can execute variety of tasks.
  - Programs written in any language.
  - Analytical/ML task → PythonOperator - sklearn
  - ETL task
  - Big Data components/programs → HiveOperator, ...
- DAGs are more dynamic, manageable, testable and collaborative.
- Airflow is job scheduler that execute the tasks on defined time and/or followed by its dependencies.

Airflow Advantages

- Dynamic DAG
  - Implemented in code - flexible
  - Can be configured: parallelism, params, templates → ninja {{ ds }}
- Extensible
  - Plenty of operators/executors
  - Shell script, Big data task, OS command, etc.
  - Custom task/plugins
- Scalable
  - Modular architecture
  - Can handle any number of DAGs (with multi-node cluster)
- Configurable
  - airflow.cfg - admin settings
  - Centralized configurations
- Monitoring
  - Elegant Web UI ৪৩৪৩
  - Handle failures (n retries)
  - Email alerts
- Open Source
  - Active community
  - New plugins

Airflow Applications

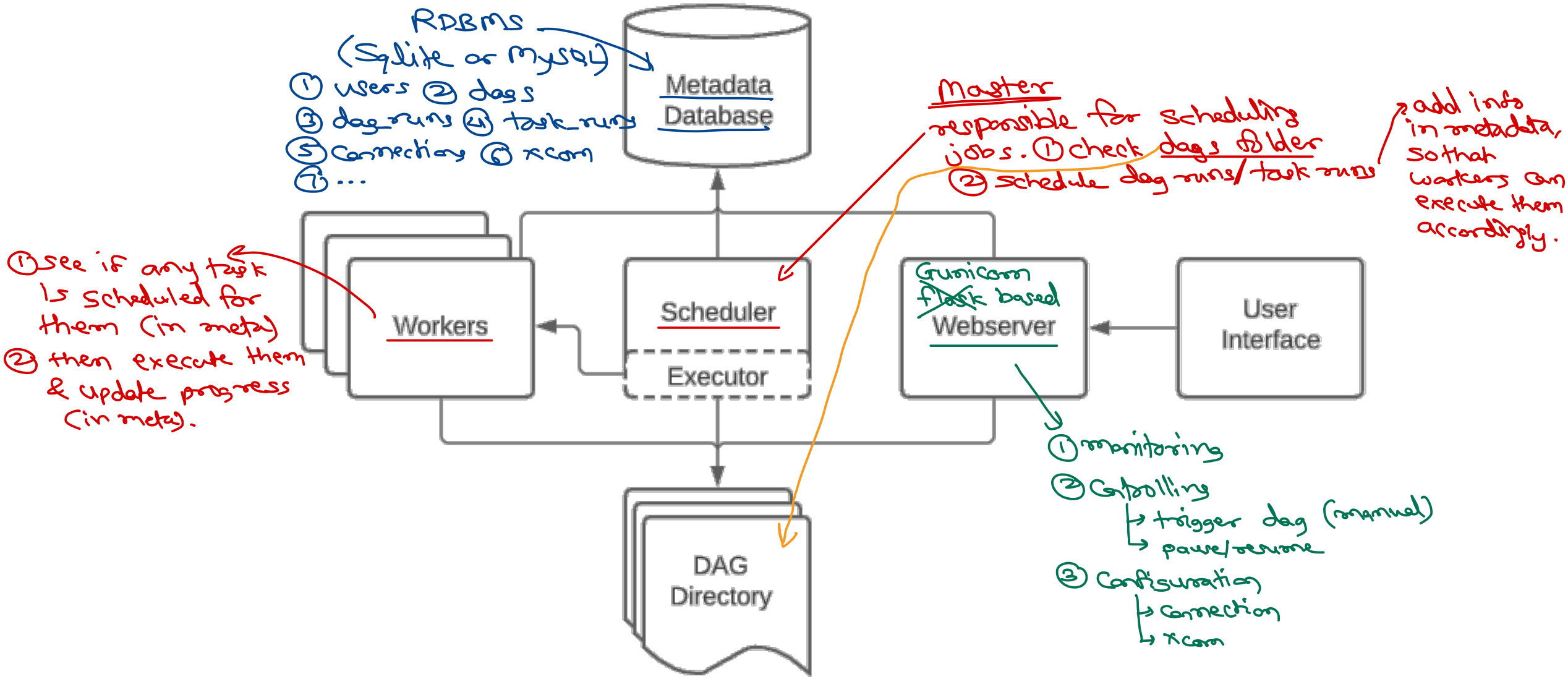
- Task scheduler
- ETL pipelines
- Periodic backups
- Automate DevOps operations
- ML jobs
- Recommendation engines
- ...

Airflow Installation

- Airflow is installed as Python package – refer docs.
- Airflow directory/files
  - config
    - airflow.cfg
      - dags folder → $AIRFLOW_HOME/dags
      - timezone → default = UTC
      - executor type → default = Sequential
      - parallelism → default = 32
  - dags → directory
    - dag definition (python files)
  - script
    - entrypoint.sh (with docker image)
    - to initiate airflow - web-server & scheduler.
    - enviroment variables
  - metastore
    - airflow.db → default = sqlite
      - using sqlite (default) database for airflow metastore.

HOME → /home/username/airflow

Airflow Architecture



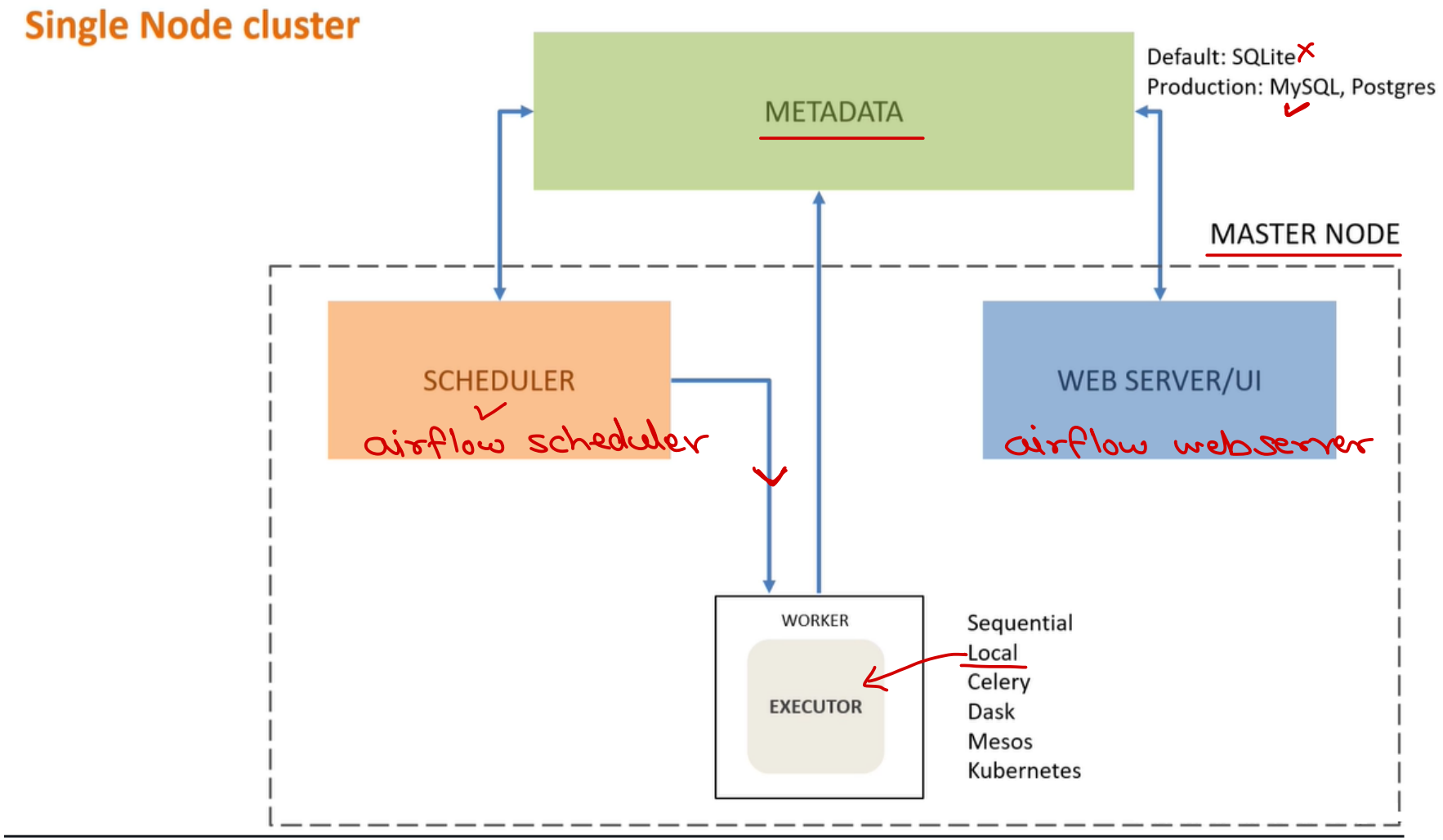
Airflow Architecture

- Metadata
  - RDBMS that stores historical and current DAGs and tasks details.
  - Also maintains information used resources.
  - Default RDBMS used is SQLite, but it is single-user database.
  - In production MySQL or Posgre-SQL is preferred option.
- Scheduler
  - Instructs and trigger task execution on worker node.
  - Implemented into Python.
  - Reads DAG file, task config and schedules the task on worker nodes in sequence.
  - It monitor tasks state from metadata and handle the task failure (as per config).
- Web Server
  - Flask-based UI for Airflow to monitor DAG.
  - Communicate with metadata to fetch the task state.
  - Task state is rendered in various formats including DAG, graphs, time/numbers, etc.

Airflow Architecture

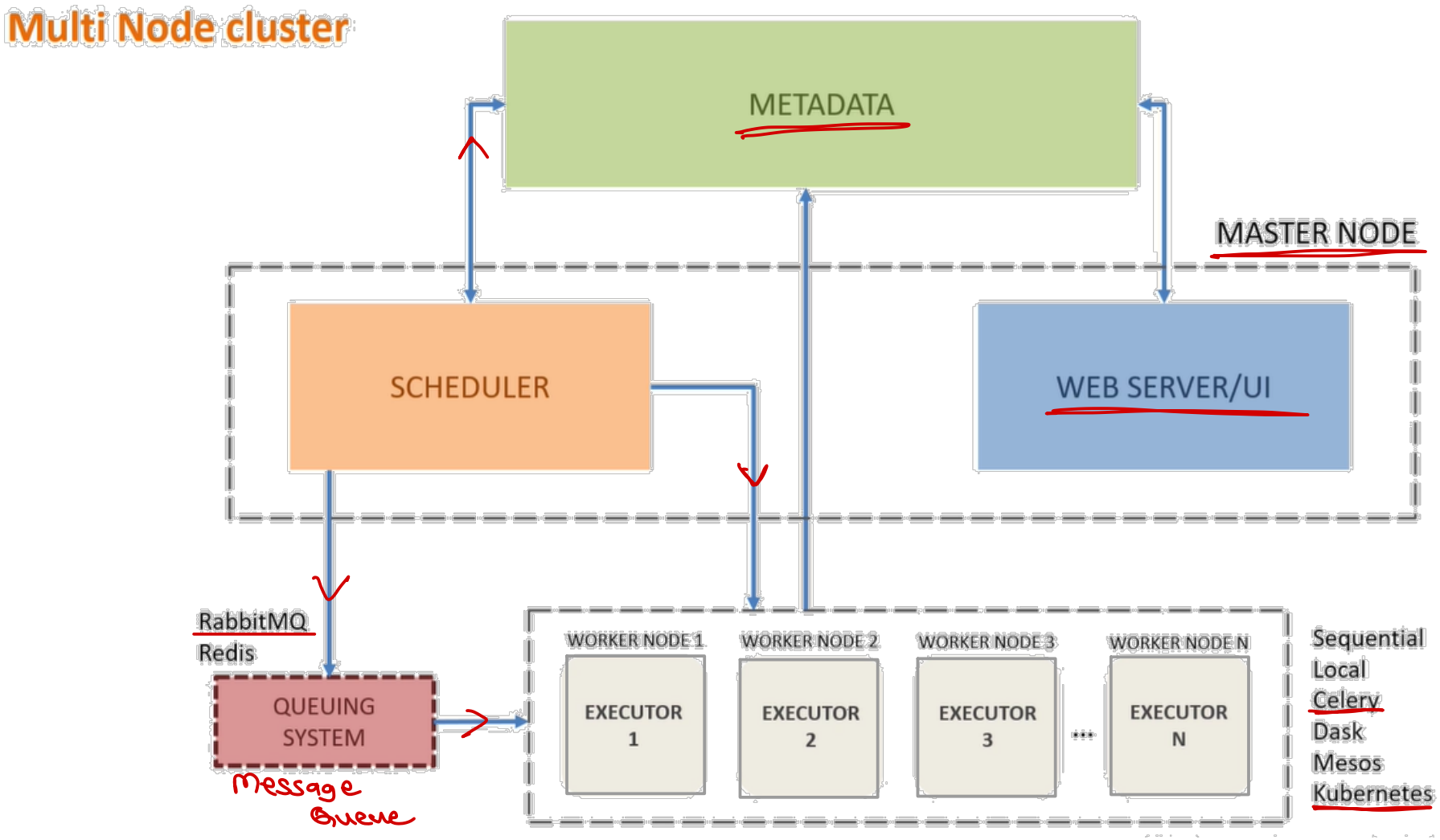
- Executor
  - Carry the task scheduled by the scheduler.
  - Runs on Worker node.
  - Various types:
    - Sequential (default) - For testing
    - Local - For single node cluster - Use python process.
    - Celery - Recommended for multi-node cluster.
      - Job queue written in Python.
      - Used in scalable distributed system.
    - Dask
    - Mesos
    - Kubernetes

Airflow – Single node installation



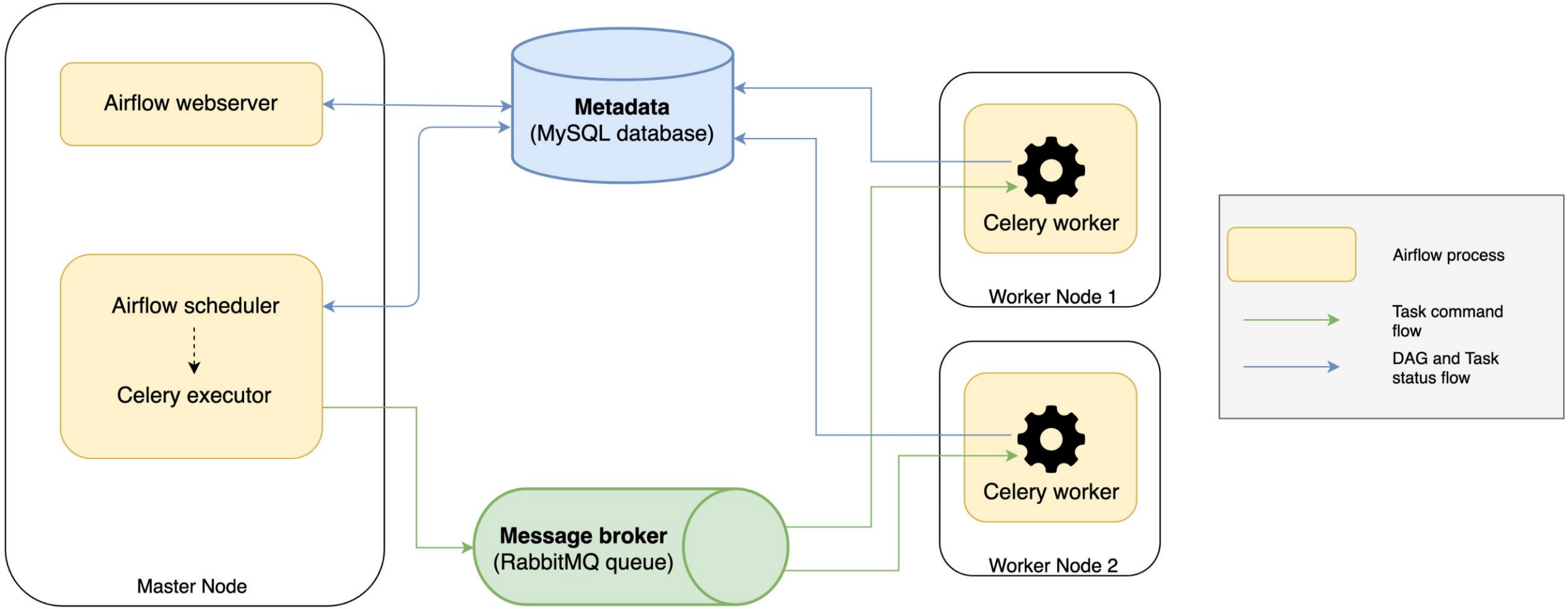
- Single worker node.
- Not scalable (max all resources available on the system).
- Quite fast for limited number of DAGs and data size.
- Use local executor.
- Direct communication between scheduler and executor.

Airflow – Multi node installation



- Master node runs scheduler and web-server.
- Multiple worker nodes - running executors.
- Highly scalable for huge data and many nodes.
- Recommended executor is Celery.
- Scheduler and workers communicate with external queue system like Rabbit MQ / Redis.

Airflow – Multi node installation



Airflow Working

- Scheduler periodically pings for DAG folder and communicate with metastore.
- If any DAG is available for execution, scheduler starts a DAG run for it. DAG run is an object representing an instantiation of DAG.
- Scheduler update DAG state as "Running".
- For each task, task object is instantiated. The task state is updated as "Scheduled".
- Scheduler assigns priority to each task in DAG (as per config) and push them into queuing system. The task state is updated to "Queued".
- Worker pull the task from queue, set its state as "Running" and start executing it.
- Upon completion of each task, task state is updated as succeed or failed.
- When all tasks in DAG run are executed, status of DAG run is updated as succeed or failed.
- Airflow web-server periodically get the data from meta-store and render it for users.
- If any new DAG is found it DAGs folder, scheduler begin its execution (as above).

Airflow Operators

- DAG only describe how to run the workflow, but do not perform actual computation.
- Operator describes single task in workflow.
- Operator characteristics
  - Atomic (usually) - standalone (not sharing resources with other operators)
  - Idempotent - produce same result for each run.
  - Operator in execution (instantiated) is referred as task.
  - Inherited from airflow's BaseOperator class.
  - Operators can communicate using XCom.
- Operator categories
  - Sensor operators - Wait for certain criteria
    - HdfsSensor, FileSensor, ...
  - Transfer operators - Transfer the data
    - MySqlToHiveOperator, ...
  - Action operators - Do specified task
    - BashOperator, PythonOperator, HiveOperator, MySqlOperator, EmailOperator, ...

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>