Bigdata Concepts

06 January 2024 09:02

7. Hadoop ETL

Lecture

- Hadoop ETL Development,
- ETL Process in Hadoop,
- Discussion of ETL functions,
- Data Extractions,
- Need of ETL tools
- Advantages of ETL tools.

Lab-Assignment:

- Understand the file formats and read the provided links

11. Apache Airflow

Lecture

- Introduction to Data Warehousing and Data Lakes
- Designing Data warehousing for an ETL Data Pipeline
- Designing Data Lakes for an ETL Data Pipeline
- ETL vs ELT
- Fundamentals of Airflow
- Work management with Airflow
- Automating an entire Data Pipeline with Airflow

Lab-Assignment:

- Create an airflow DAG for Extract -> Transform -> Load

Hadoop ETL (Extract , Transform and Load)

- Hadoop is often used in ETL process for handling large-scale data processing
- ETL development in Hadoop - involves extracting data from various sources, transform it to meet specific requirements and loading it into a target data store or system.

1. Data Extraction --> we need use Hadoop compatible tools and frameworks -> Apache Sqoop, Apache Nifi, Custom MapReduce / Spark jobs to ingest data into the HDFS. Extracted data could be form RDBMS, log files, APIs, or other sources.

2. Data Transformation --> Desing and implement data transformation logic to clean, enrich and structure data as per business requirement.
   a. We utilize Hadoop processing framework such as Apache Hive, Apache Pig, Apace Spark or Custom MapReduce jobs to perform transformations.
   b. Transformation --> filtering, aggregating, joining and apply business rules to the data.
   c. Eg: in hive tables format --> csv, orc, avro (json) , sequence, parquet

3. Data Quality and Validation
   a. Check and validate process to ensure the integrity and reliability of the transformed data
   b. Identify and handle missing or erroneous data during the transformation phase.
   c. Tools used --> Apache Hadoop MapReduce or Apache Spark to execute custom validation logic

4. Data loading --> decide on the target data store or system for loading the transformed data.
   a. This could be Hadoop based storage (eg: Hive, Hbase) or any other external data warehouse
   b. Utilize Hadoop-compatible tools or custom scripts to load the transformed data into the target system.
   c. Consider optimizing data loading for performance when dealing with large datasets.

5. Job scheduling and orchestration:
   a. To automate the execution of ETL workflows we can use Apache Airflow, or job scheduling tools like Apache Oozie
   b. Scheduling ETL jobs to run at specific intervals or trigger them based on events such as data arrival or changes in source system.

6. Monitoring and Logging
   a. Implement monitoring and Logging mechanisms to track the progress and health of ETL jobs.
   b. Hadoop echo system tools for monitoring --> Apache Ambari or Cloudera Manger
   c. Keep track on --> resource utilization, job completion status and overall system health.
   d. Two type of monitoring -> proactive (preferred) and reactive

7. Error Handling and Recovery
   a. Implement error handling mechanism to gracefully handle all failures during the ETL process
   b. Data recovery strategies --> rerunning failed tasks or rolling back incomplete transactions

8. Metadata Management
   a. Maintain metadata to track the lineage and characteristics of the data through the ETL process.
   b. Tools like Apache Atlas - used for metadata management and data governance.

9. Security and Access Control
    a. Implement appropriate security measures to protect all type of sensitive data
    b. We can make use of Hadoop's security features, include Kerberos Authentication and ACL's (Access Control List) { RBAC - Role based access control list), to control access to data.

10. Performance Tuning
    a. Optimize ETL jobs for performance by considering --> resource utilization, parallel processing , data partition
    b. Monitor and tune the performance of transformation to ensure efficient processing

► Creating an table in hive which used both bucketing and partitioning

Eg: for bucketing

```
create table newstore (
storeid int,
storename string,
storeprice float,
storetype string
) CLUSTERED By (storeid) into 2  Buckets ;
```

Eg: for partitioning

```
create table part_table1 (
 slnno int,
  name string,
  city string,
  location string,
   zipcodes int)
   PARTITIONED BY(state string)
   ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';
```

Combining Both Partitioning and Bucketing:

Eg:

```
Create table bucket_part_tbl (
Slno int,
Name string,
Location string,
Storename string,
Storetype string,
Zicodes int )
Partitioned by (state string)
Clustered by (Storename) into 4 buckets;
```

```
hive> insert into table bucket_part_tbl partition (state='karnatka') values (1,'giridhar','bangalore', 'bookstore', 'stationery',560001);
hive>insert into table bucket_part_tbl partition (state='mumbai') values (2,'suri','mumbai central','knife and scissors' ,'kitchen items',660001);
hive> insert into table bucket_part_tbl partition (state='chennai') values (3,'vadivel','cennai centrol','idly and sambar','hotels', 760002);
hive>insert into table bucket_part_tbl partition (state='apandtg') values (4,'balaji','tirupathi', 'tirumala', 'temple', 160021);
hive> insert into table bucket_part_tbl partition (state='kerala') values (5,'kk nair','triuvanathpuram','temple treasure','godsowncountry', 450001);
```

```
$ hdfs  dfs -ls   /user/hive/warehouse/sampledb.db/bucket_part_tbl
```

You can view the partitioned data based upon **"state"** when you query your hdfs as shown below

```
hadoop@mainserver1:~$ hdfs dfs -ls /user/hive/warehouse/sampledb.db/bucket_part_tbl
2024-01-06 06:23:50,777 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classe
s where applicable
Found 5 items
drwxr-xr-x   - hadoop supergroup          0 2024-01-06 06:18 /user/hive/warehouse/sampledb.db/bucket_part_tbl/state=apandtg
drwxr-xr-x   - hadoop supergroup          0 2024-01-06 06:18 /user/hive/warehouse/sampledb.db/bucket_part_tbl/state=chennai
drwxr-xr-x   - hadoop supergroup          0 2024-01-06 06:18 /user/hive/warehouse/sampledb.db/bucket_part_tbl/state=karnatka
drwxr-xr-x   - hadoop supergroup          0 2024-01-06 06:18 /user/hive/warehouse/sampledb.db/bucket_part_tbl/state=kerala
drwxr-xr-x   - hadoop supergroup          0 2024-01-06 06:18 /user/hive/warehouse/sampledb.db/bucket_part_tbl/state=mumbai
hadoop@mainserver1:~$
```

You can view the buckets of **"4 numbers"** created when you query your hdfs as shown below

```
hadoop@mainserver1:~$ hdfs dfs -ls /user/hive/warehouse/sampledb.db/bucket_part_tbl/state=karnatka
2024-01-06 06:27:26,689 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-java classe
s where applicable
Found 4 items
-rwxr-xr-x   1 hadoop supergroup         49 2024-01-06 06:18 /user/hive/warehouse/sampledb.db/bucket_part_tbl/state=karnatka/000000_0
-rwxr-xr-x   1 hadoop supergroup          0 2024-01-06 06:18 /user/hive/warehouse/sampledb.db/bucket_part_tbl/state=karnatka/000001_0
-rwxr-xr-x   1 hadoop supergroup          0 2024-01-06 06:18 /user/hive/warehouse/sampledb.db/bucket_part_tbl/state=karnatka/000002_0
-rwxr-xr-x   1 hadoop supergroup          0 2024-01-06 06:18 /user/hive/warehouse/sampledb.db/bucket_part_tbl/state=karnatka/000003_0
hadoop@mainserver1:~$
```

```
hive> select * from  bucket_part_tbl;
OK
4       balaji  tirupathi       tirumala        temple  160021  apandtg
3       vadivel cennai centrol  idly and sambar hotels  760002  chennai
1       giridhar        bangalore       bookstore       stationery      560001  karnatka
5       kk nair triuvanathpuram temple treasure godsowncountry  450001  kerala
2       suri    mumbai central  knife and scissors      kitchen items   660001  mumbai
Time taken: 1.521 seconds, Fetched: 5 row(s)
hive>
```

```
hive> describe bucket_part_tbl;
OK
slno                    int
name                    string
location                string
storename               string
storetype               string
zicodes                 int
state                   string

# Partition Information
# col_name              data_type               comment

state                   string
Time taken: 1.04 seconds, Fetched: 12 row(s)
hive>
```

Sample Example of ETL: Logfiles stored in a rational database and performing ETL to analyze user activities

1. Extraction ---> use Apache Sqoop to import log data from the relational data base to HDFS
2. Transformation --> use Apache Spark to process log data, extract relevant info, aggregate metric and enrich data as needed
3. Quality and validity --> Implement spark jobs or custom mapreduce tasks {map only or reduce only), perform data quality checks, verify inconsistent data or handle missing information
4. Loading --> load the transformed data to Apache hive tables or other storage systems for further analysis
5. Job scheduling and orchestration ---> Apache Oozie to run daily schedules as per the ETL workflow , process the latest log data
6. Monitoring and logging ---> Apache Log4j to setup logging and Apache Ambari or Cloudera manager for monitoring jobs
7. Error handling and recovery --> implement error-handling mechanisms to retry failed tasks and recovery form errors during the ETL process , depends up the tools used.
8. Metadata management --> user Apache Atlas - tracking the lineage of data from source to destination
9. Security and access control --> depends up the work flow - we can use Kerberos authentication
10. Performance tuning --> we can optimize Apache Spark configuration, adjust the number of executors, and partition data as per the requirements And hence ensure efficient ETL processing take place.

(3A's --> Authentication, Authorization and Auditing)

Need or Advantages of ETL tools

1. Data integration and consolidation : -Since Hadoop deals with structured as well as unstructured data, hence diverse data sources are involved ETL tools helps in integrating and consolidating data from various sources into a centralized storge systems like HDFS
2. Apache pig, hive and spark etc., are some of the tools designed to work seemlessly with Hadoop echo systems for ETL workflows. And hence helps in using the power of distributed computing for data processing
3. Integration and support for traditional data warehouse
4. Provides scalability -- Hadoop provides horizontal scalability and we can leverage the use of parallel processing.
5. Ease of user --> provides user friendly interfaces and visual design capabilities.
6. Data transformation to convert raw data into a format suitable for analysis
7. We can manage complex data processing task efficiently --> distributing and parallel data processing across Hadoop clusters
8. Cost-effective storage --> allows us to store and process data as scale without incurring significant storage cost
9. Ensures sensitive data is handled securely and only authorized uses have access to specific data sets

ETL Process in Hadoop

The ETL (Extract, Transform, Load) process in Hadoop involves extracting data from various sources, transforming it into the desired format, and loading it into a target system. Hadoop provides a distributed and scalable platform for handling large-scale ETL workloads. Here's a high-level overview of the ETL process in Hadoop:

1. **Data Extraction:**

Sources:

- Databases: Extract data from relational databases using tools like Apache Sqoop or custom MapReduce/Spark jobs.
- Log Files: Ingest data from log files stored in HDFS or other distributed file systems.
- External APIs: Fetch data from external APIs using custom scripts or tools like Apache NiFi.
- Other Data Sources: Extract data from various sources, such as cloud storage, NoSQL databases, or streaming platforms.

Hadoop Tools:

- Sqoop: Import data from relational databases into HDFS.
- Flume or Kafka: Ingest data from log files or streaming sources.
- Custom MapReduce or Spark Jobs: Develop custom scripts to perform extraction tasks.

2. **Data Transformation:**
Processing Frameworks:
- Apache Spark: Use Spark for batch processing or stream processing to transform data at scale.
- Apache Hive: Utilize Hive for SQL-based transformations on structured data.
- Apache Pig: Write Pig Latin scripts for data transformation tasks.
- Custom MapReduce Jobs: Develop custom MapReduce programs for specific transformation requirements.

Transformation Tasks:

- Filtering and Cleansing: Remove or handle missing and erroneous data.
- Aggregation: Aggregate data based on certain dimensions or criteria.
- Enrichment: Add additional information to the data.
- Joining: Combine data from multiple sources.
- Format Conversion: Convert data into the desired format.

3. **Data Quality and Validation:**
- Implement checks to ensure data quality and consistency.
- Handle missing or invalid data gracefully.
- Log errors and exceptions for later analysis.
- Utilize custom MapReduce jobs, Spark, or other processing frameworks for validation tasks.

4. **Data Loading:**
Targets:
- HDFS: Load transformed data back into HDFS for further processing.
- Apache Hive or HBase: Store structured data in Hive tables or HBase for analytical queries.
- External Data Warehouses: Load data into external data warehouses like Amazon Redshift or Google BigQuery.
- Data Lakes: Populate data lakes with processed data.

Loading Mechanisms:
- Apache Sqoop: Export data from HDFS to relational databases.
- Custom MapReduce or Spark Jobs: Develop custom loading scripts for specific data targets.
- Hive or Pig Scripts: Use Hive or Pig to insert transformed data into target tables.

5. **Job Scheduling and Orchestration:**
- Use workflow management tools like Apache Oozie or Apache Airflow to schedule and coordinate ETL jobs.
- Schedule jobs at specific intervals or trigger them based on external events.

6. **Monitoring and Logging:**
- Implement logging mechanisms to capture information about job execution, errors, and warnings.
- Utilize monitoring tools like Apache Ambari, Cloudera Manager, or custom solutions to track the progress and health of ETL jobs.

7. **Error Handling and Recovery:**
- Implement mechanisms to handle errors during the ETL process.
- Retry failed tasks or take appropriate actions based on the nature of errors.
- Design recovery strategies to ensure the ETL process can resume from a consistent state.

8. **Metadata Management:**
- Maintain metadata to track the lineage of data from source to destination.
- Use tools like Apache Atlas for metadata management and data governance.

9. **Security and Access Control:**
- Apply security measures to protect sensitive data during the ETL process.
- Utilize features like Kerberos authentication and Access Control Lists (ACLs) in Hadoop.

10. **Performance Tuning:**
- Optimize ETL jobs for performance by considering factors such as data partitioning, parallel processing, and resource utilization.
- Monitor and tune the performance of transformations to ensure efficient processing.

Example Scenario:
Consider a scenario where you want to perform ETL on web server log data stored in HDFS:

1. Extraction:
   - Use Apache Flume or a custom script to ingest log data into HDFS.
2. Transformation:
   - Utilize Apache Spark to parse log entries, extract relevant information, and aggregate metrics such as page views or unique visitors.
3. Data Quality and Validation:
   - Implement Spark jobs to validate log entries, handle missing data, and identify anomalies.
4. Loading:
   - Load the transformed data into Apache Hive tables for analysis.
5. Job Scheduling and Orchestration:
   - Schedule the ETL workflow using Apache Oozie to run hourly, processing the latest log data.
6. Monitoring and Logging:
   - Implement logging using tools like Apache Log4j and monitor job progress using Apache Ambari.
7. Error Handling and Recovery:
   - Design error-handling mechanisms to handle data anomalies and rerun failed tasks if necessary.
8. Metadata Management:
   - Use Apache Atlas to maintain metadata, tracking the lineage of data from log sources to Hive tables.
9. Security and Access Control:
   - Apply security measures such as Kerberos authentication to secure the ETL process.
10. Performance Tuning:
    - Optimize Spark configurations and adjust the number of executors to ensure efficient processing of log data.

Remember that the specific tools, frameworks, and steps used in ETL development may vary based on your organization's preferences, data sources, and requirements. Always adapt the ETL process to the characteristics of your specific use case.

ETL functions

ETL (Extract, Transform, Load) functions are essential processes in data management and analytics workflows. These functions are crucial for moving, processing, and preparing data for analysis. Let's discuss each ETL function in detail:

1. Extract:
   - Purpose: Extracting data from source systems, which can be databases, flat files, APIs, or other data repositories.
   - Tasks:
     - Reading data from source systems.
     - Extracting relevant data based on criteria.
     - Handling incremental data extraction to only retrieve new or updated records.
   - Tools/Technologies:
     - Apache Sqoop, Apache NiFi, Talend, Informatica, DataStage.
2. Transform:
   - Purpose: Transforming raw data into a suitable format for analysis, reporting, or storage.
   - Tasks:
     - Cleaning and validating data.
     - Aggregating and summarizing data.
     - Handling data enrichment by combining data from different sources.
     - Converting data types and formats.
     - Handling missing or erroneous data.
   - Tools/Technologies:
     - Apache Spark, Apache Hive, Apache Pig, Talend, Informatica, DataStage.
3. Load:
   - Purpose: Loading transformed data into a target system, often a data warehouse or database.
   - Tasks:
     - Defining the structure of the target data store.
     - Loading data into tables or data structures.
     - Managing data partitioning and indexing for performance.
     - Handling data integrity and constraints.
     - Logging and monitoring load processes.
   - Tools/Technologies:
     - Apache Hive, Apache HBase, Apache Cassandra, Talend, Informatica, DataStage.
4. Data Integration:
   - Purpose: Combining data from various sources to provide a unified view.
   - Tasks:
     - Merging data from different databases or systems.
     - Resolving conflicts and duplicates.
     - Creating a cohesive dataset for analysis.
   - Tools/Technologies:
     - Apache NiFi, Talend, Informatica, DataStage.
5. Data Quality Management:
   - Purpose: Ensuring the accuracy, consistency, and reliability of data.
   - Tasks:

    - Data profiling to identify anomalies.
    - Implementing validation rules.
    - Cleansing and standardizing data.
    - Handling error detection and correction.
  - Tools/Technologies:
    - Talend, Informatica, Data Quality Tools.
6. Workflow Orchestration:
  - Purpose: Managing and orchestrating the sequence of ETL tasks.
  - Tasks:
    - Designing and scheduling ETL workflows.
    - Managing dependencies between tasks.
    - Monitoring and logging job executions.
  - Tools/Technologies:
    - Apache Oozie, Apache Airflow, Talend, Informatica.
7. Schema Evolution:
  - Purpose: Adapting to changes in data structures over time.
  - Tasks:
    - Modifying data models and schemas.
    - Handling backward and forward compatibility.
    - Ensuring data consistency during schema changes.
  - Tools/Technologies:
    - Schema Evolution in Hive, Avro, Parquet.
8. Data Partitioning:
  - Purpose: Organizing data into partitions for better query performance.
  - Tasks:
    - Dividing data into logical partitions based on specific columns.
    - Improving query efficiency by limiting the data scanned.
    - Managing partitioning strategies.
  - Tools/Technologies:
    - Apache Hive, Apache Impala, Apache Spark.
9. Incremental Data Processing:
  - Purpose: Handling updates and additions to existing data.
  - Tasks:
    - Identifying and processing only new or modified records.
    - Managing watermarking or timestamp-based incremental loads.
    - Ensuring data consistency during incremental updates.
  - Tools/Technologies:
    - Apache Sqoop, Apache NiFi, Change Data Capture (CDC) tools.
10. Metadata Management:
  - Purpose: Managing and documenting the metadata associated with data sources, transformations, and loading processes.
  - Tasks:
    - Capturing and documenting metadata information.
    - Tracking lineage and dependencies.
    - Enabling data governance and compliance.
  - Tools/Technologies:
    - Apache Atlas, Collibra, Talend, Informatica.

Conclusion :

ETL functions are crucial for maintaining data quality, ensuring data consistency, and providing a solid foundation for analytics and business intelligence in modern data-driven organizations. The choice of specific tools and technologies depends on the organization's requirements, existing infrastructure, and the complexity of data integration and processing tasks.

Apache Airflow Installation Process

Steps to Install and Setup Apache Airflow on Ubuntu 20.04 LTS

Apache-Airflow is a free & open source workflow management tool, written in Python. It is used for monitoring the workflow & is a workflow management solutions tool. We can easily visualize the data pipelines' dependencies, progress, logs, code, trigger tasks, and success status.

Update the System.

```
apt-get update
```

Install the required packages.

```
apt-get install software-properties-common
apt-add-repository universe
```

- Update the packages.

```
apt-get update
```

Install the python3-pip.

```
apt-get install python-setuptools
apt install python3-pip
```

Install the required dependencies for Apache Airflow.

```
apt-get install libmysqlclient-dev
apt-get install libssl-dev
apt-get install libkrb5-dev
```

Install the Apache-Airflow on system.

- Install the python-virtual Environment.

```
apt install python3-virtualenv
virtualenv airflow_example
```

- Change the directory.

```
cd airflow_example/bin/
```

- Activate the source.

```
source activate
```

- Now ,install the apache-airflow.

```
export AIRFLOW_HOME=~/airflow
pip3 install apache-airflow
```

- Install typing_extension.

```
pip3 install typing_extensions
```

- Run the following command.

```
airflow db init
```

Set the Apache-Airflow login credentials for airflow web interface.

```
airflow users create --username admin --firstname FIRST_NAME --lastname  LAST_NAME --role Admin --email admin@example.org
```

For example:

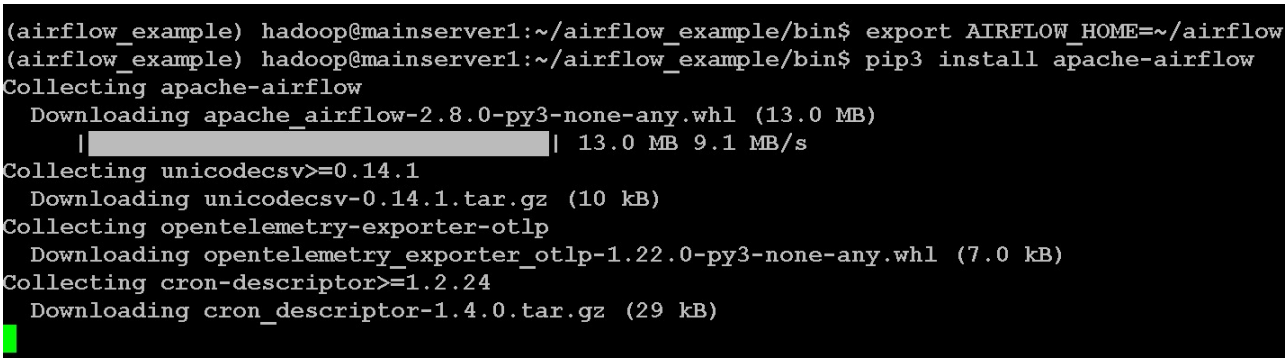
```
airflow users create --username admin --firstname admin --lastname testing --role Admin --email admin@domain.com
```

Using nohup, start the Airflow scheduler to run in the background. Airflow appends the output of running the scheduler to the scheduler.log file.

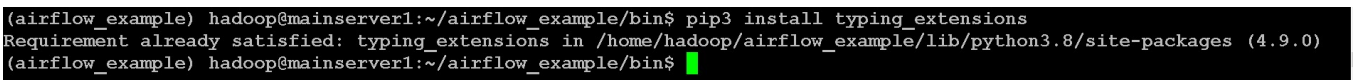
```
$ nohup airflow scheduler > scheduler.log 2>&1 &
```

Start the Apache-Airflow web interface.

```
$ nohup airflow webserver -p 8080 > webserver.log  2>&1 &
```



After completion of the above command run this



```
(airflow_example) hadoop@mainserver1:~/airflow_example/bin$ airflow db init
/home/hadoop/airflow_example/lib/python3.8/site-packages/airflow/cli/commands/db_command.py:47 DeprecationWarning: `db
 init` is deprecated.  Use `db migrate` instead to migrate the db and/or airflow connections create-default-connection
s to create the default connections
DB: sqlite:////home/hadoop/airflow/airflow.db
[2024-01-06T09:58:03.441+0000] {migration.py:216} INFO - Context impl SQLiteImpl.
[2024-01-06T09:58:03.443+0000] {migration.py:219} INFO - Will assume non-transactional DDL.
INFO  [alembic.runtime.migration] Context impl SQLiteImpl.
INFO  [alembic.runtime.migration] Will assume non-transactional DDL.
INFO  [alembic.runtime.migration] Running stamp_revision  -> 10b52ebd31f7
WARNI [airflow.models.crypto] empty cryptography key - values will not be stored encrypted.
Initialization done
(airflow_example) hadoop@mainserver1:~/airflow_example/bin$
```

$ airflow users create --username admin --firstname FIRST_NAME --lastname LAST_NAME --role Admin --email admin@example.org

```
(airflow_example) hadoop@mainserver1:~/airflow_example/bin$ airflow users create --username admin --firstname FIRST_NA
ME --lastname  LAST_NAME --role Admin --email admin@example.org
/home/hadoop/airflow_example/lib/python3.8/site-packages/flask_limiter/extension.py:336 UserWarning: Using the in-memo
ry storage for tracking rate limits as no storage was explicitly specified. This is not recommended for production use
. See: https://flask-limiter.readthedocs.io#configuring-a-storage-backend for documentation about configuring the stor
age backend.
[2024-01-06T10:00:58.562+0000] {override.py:1369} INFO - Inserted Role: Admin
[2024-01-06T10:00:58.583+0000] {override.py:1369} INFO - Inserted Role: Public
[2024-01-06T10:00:58.598+0000] {override.py:868} WARNING - No user yet created, use flask fab command to do it.
[2024-01-06T10:00:58.744+0000] {override.py:1769} INFO - Created Permission View: can edit on Passwords
[2024-01-06T10:00:58.767+0000] {override.py:1820} INFO - Added Permission can edit on Passwords to role Admin
[2024-01-06T10:00:58.802+0000] {override.py:1769} INFO - Created Permission View: can read on Passwords
[2024-01-06T10:00:58.822+0000] {override.py:1820} INFO - Added Permission can read on Passwords to role Admin
[2024-01-06T10:00:58.880+0000] {override.py:1769} INFO - Created Permission View: can edit on My Password
[2024-01-06T10:00:58.899+0000] {override.py:1820} INFO - Added Permission can edit on My Password to role Admin
[2024-01-06T10:00:58.924+0000] {override.py:1769} INFO - Created Permission View: can read on My Password
[2024-01-06T10:00:58.945+0000] {override.py:1820} INFO - Added Permission can read on My Password to role Admin
[2024-01-06T10:00:58.988+0000] {override.py:1769} INFO - Created Permission View: can edit on My Profile
[2024-01-06T10:00:59.008+0000] {override.py:1820} INFO - Added Permission can edit on My Profile to role Admin
[2024-01-06T10:00:59.035+0000] {override.py:1769} INFO - Created Permission View: can read on My Profile
[2024-01-06T10:00:59.058+0000] {override.py:1820} INFO - Added Permission can read on My Profile to role Admin
[2024-01-06T10:00:59.203+0000] {override.py:1769} INFO - Created Permission View: can create on Users
[2024-01-06T10:00:59.219+0000] {override.py:1820} INFO - Added Permission can create on Users to role Admin
[2024-01-06T10:00:59.245+0000] {override.py:1769} INFO - Created Permission View: can read on Users
[2024-01-06T10:00:59.263+0000] {override.py:1820} INFO - Added Permission can read on Users to role Admin
[2024-01-06T10:00:59.285+0000] {override.py:1769} INFO - Created Permission View: can edit on Users
[2024-01-06T10:00:59.315+0000] {override.py:1820} INFO - Added Permission can edit on Users to role Admin
[2024-01-06T10:00:59.349+0000] {override.py:1769} INFO - Created Permission View: can delete on Users
[2024-01-06T10:00:59.384+0000] {override.py:1820} INFO - Added Permission can delete on Users to role Admin
[2024-01-06T10:00:59.451+0000] {override.py:1769} INFO - Created Permission View: menu access on List Users
```

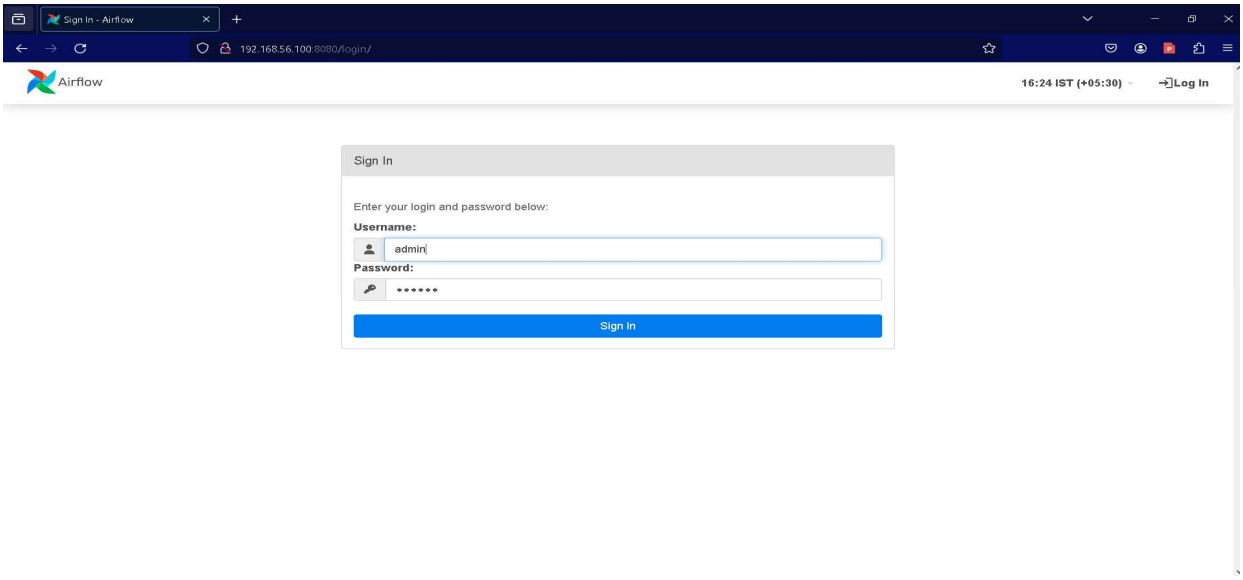
```
root@mainserver1: ~
[2024-01-06T10:00:59.643+0000] {override.py:1820} INFO - Added Permission can create on Roles to role Admin
[2024-01-06T10:00:59.666+0000] {override.py:1769} INFO - Created Permission View: can read on Roles
[2024-01-06T10:00:59.689+0000] {override.py:1820} INFO - Added Permission can read on Roles to role Admin
[2024-01-06T10:00:59.717+0000] {override.py:1769} INFO - Created Permission View: can edit on Roles
[2024-01-06T10:00:59.767+0000] {override.py:1820} INFO - Added Permission can edit on Roles to role Admin
[2024-01-06T10:00:59.807+0000] {override.py:1769} INFO - Created Permission View: can delete on Roles
[2024-01-06T10:00:59.830+0000] {override.py:1820} INFO - Added Permission can delete on Roles to role Admin
[2024-01-06T10:00:59.871+0000] {override.py:1769} INFO - Created Permission View: menu access on List Roles
[2024-01-06T10:00:59.890+0000] {override.py:1820} INFO - Added Permission menu access on List Roles to role Admin
[2024-01-06T10:00:59.960+0000] {override.py:1769} INFO - Created Permission View: can read on User Stats Chart
[2024-01-06T10:00:59.980+0000] {override.py:1820} INFO - Added Permission can read on User Stats Chart to role Admin
[2024-01-06T10:01:00.031+0000] {override.py:1769} INFO - Created Permission View: menu access on User's Statistics
[2024-01-06T10:01:00.051+0000] {override.py:1820} INFO - Added Permission menu access on User's Statistics to role Adm
in
[2024-01-06T10:01:00.163+0000] {override.py:1769} INFO - Created Permission View: can read on Permissions
[2024-01-06T10:01:00.183+0000] {override.py:1820} INFO - Added Permission can read on Permissions to role Admin
[2024-01-06T10:01:00.230+0000] {override.py:1769} INFO - Created Permission View: menu access on Actions
[2024-01-06T10:01:00.249+0000] {override.py:1820} INFO - Added Permission menu access on Actions to role Admin
[2024-01-06T10:01:00.357+0000] {override.py:1769} INFO - Created Permission View: can read on View Menus
[2024-01-06T10:01:00.377+0000] {override.py:1820} INFO - Added Permission can read on View Menus to role Admin
[2024-01-06T10:01:00.431+0000] {override.py:1769} INFO - Created Permission View: menu access on Resources
[2024-01-06T10:01:00.456+0000] {override.py:1820} INFO - Added Permission menu access on Resources to role Admin
[2024-01-06T10:01:00.551+0000] {override.py:1769} INFO - Created Permission View: can read on Permission Views
[2024-01-06T10:01:00.570+0000] {override.py:1820} INFO - Added Permission can read on Permission Views to role Admin
[2024-01-06T10:01:00.620+0000] {override.py:1769} INFO - Created Permission View: menu access on Permission Pairs
[2024-01-06T10:01:00.643+0000] {override.py:1820} INFO - Added Permission menu access on Permission Pairs to role Admi
n
Password:
Repeat for confirmation:
[2024-01-06T10:01:17.139+0000] {override.py:1458} INFO - Added user admin
User "admin" created with role "Admin"
(airflow_example) hadoop@mainserver1:~/airflow_example/bin$
```

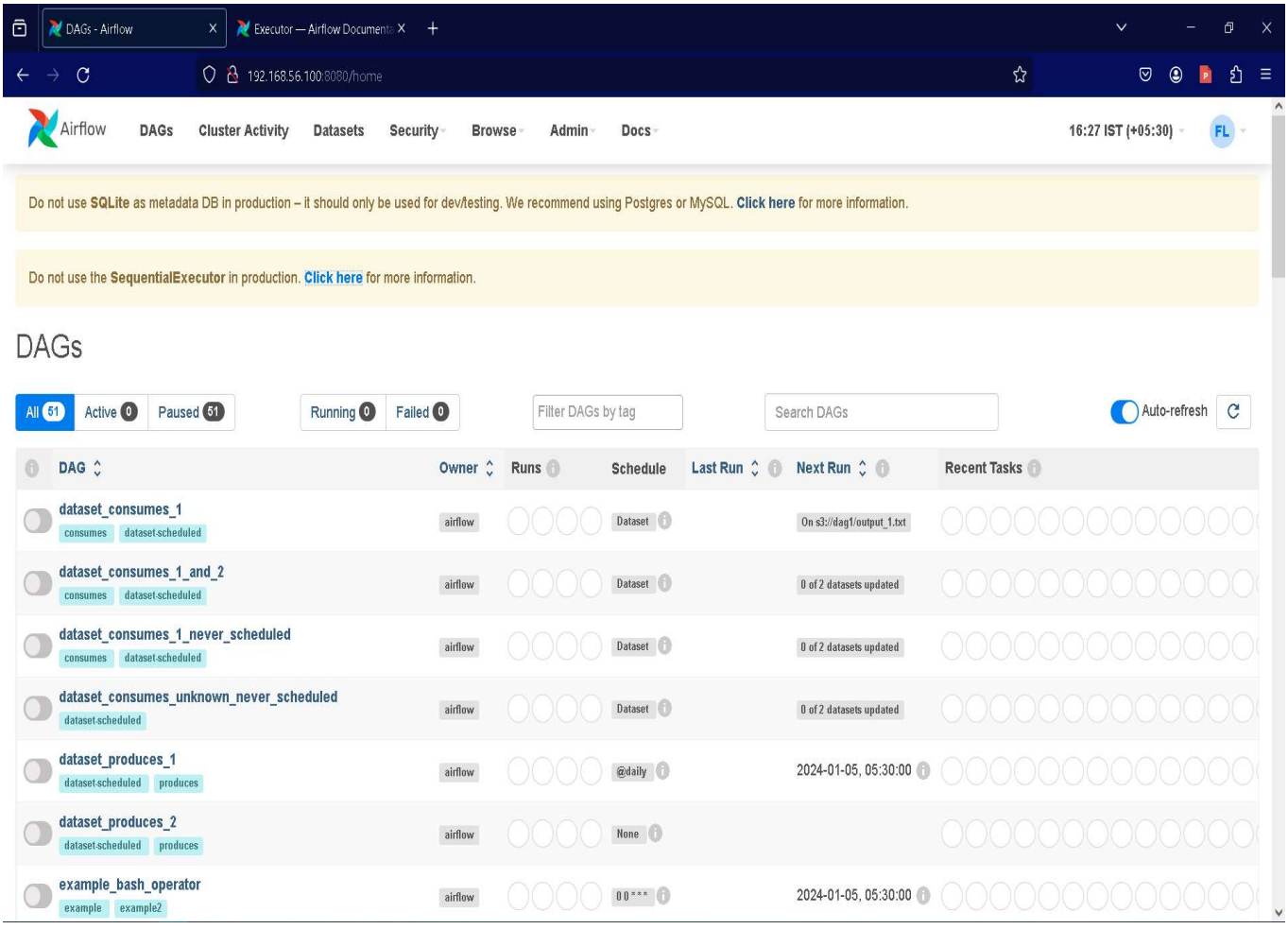
```
(airflow_example) hadoop@mainserver1:~/airflow_example/bin$ nohup airflow scheduler > scheduler.log 2>&1 &
[1] 14976
(airflow_example) hadoop@mainserver1:~/airflow_example/bin$ ps -a
    PID TTY          TIME CMD
   1035 tty1     00:00:00 bash
  14976 pts/1    00:00:27 airflow
  14979 pts/1    00:00:00 gunicorn: maste
  14980 pts/1    00:00:00 gunicorn: worke
  14981 pts/1    00:00:10 airflow schedul
  14982 pts/1    00:00:00 gunicorn: worke
  15096 pts/1    00:00:00 ps
(airflow_example) hadoop@mainserver1:~/airflow_example/bin$
```

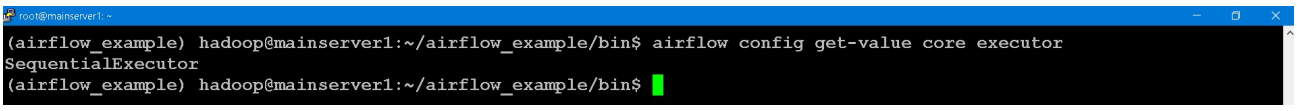
```
(airflow_example) hadoop@mainserver1:~/airflow_example/bin$ nohup airflow webserver -p 8080 > webserver.log  2>&1 &
[2] 15303
```

```
(airflow_example) hadoop@mainserver1:~/airflow_example/bin$ ps -a
    PID TTY          TIME CMD
   1035 tty1     00:00:00 bash
  14976 pts/1    00:01:20 airflow
  14979 pts/1    00:00:00 gunicorn: maste
  14980 pts/1    00:00:00 gunicorn: worke
  14981 pts/1    00:00:33 airflow schedul
  14982 pts/1    00:00:00 gunicorn: worke
  15303 pts/1    00:00:19 airflow
  15311 pts/1    00:00:13 gunicorn: maste
  15326 pts/1    00:00:00 [ready] gunicor
  15327 pts/1    00:00:00 [ready] gunicor
  15328 pts/1    00:00:00 [ready] gunicor
  15329 pts/1    00:00:00 [ready] gunicor
  15384 pts/1    00:00:00 ps
(airflow_example) hadoop@mainserver1:~/airflow_example/bin$
```

```
(airflow_example) hadoop@mainserver1:~/airflow_example/bin$ ss -ant
State          Recv-Q      Send-Q            Local Address:Port             Peer Address:Port      Process
LISTEN         0           128                     0.0.0.0:41369                 0.0.0.0:*
LISTEN         0           128                     0.0.0.0:13562                 0.0.0.0:*
LISTEN         0           128                     0.0.0.0:8030                  0.0.0.0:*
LISTEN         0           128                     0.0.0.0:8031                  0.0.0.0:*
LISTEN         0           128                   127.0.0.1:36767                 0.0.0.0:*
LISTEN         0           128                     0.0.0.0:8032                  0.0.0.0:*
LISTEN         0           128                     0.0.0.0:8033                  0.0.0.0:*
LISTEN         0           128                     0.0.0.0:8040                  0.0.0.0:*
LISTEN         0           4096                    0.0.0.0:9864                  0.0.0.0:*
LISTEN         0           128                   127.0.0.1:9000                  0.0.0.0:*
LISTEN         0           128                     0.0.0.0:8042                  0.0.0.0:*
LISTEN         0           128                     0.0.0.0:9866                  0.0.0.0:*
LISTEN         0           128                     0.0.0.0:9867                  0.0.0.0:*
LISTEN         0           128                     0.0.0.0:9868                  0.0.0.0:*
LISTEN         0           128                     0.0.0.0:9870                  0.0.0.0:*
LISTEN         0           2048                    0.0.0.0:8080                  0.0.0.0:*
LISTEN         0           4096               127.0.0.53%lo:53                   0.0.0.0:*
LISTEN         0           128                     0.0.0.0:22                    0.0.0.0:*
LISTEN         0           128                     0.0.0.0:8088                  0.0.0.0:*
ESTAB          0           0                     127.0.0.1:42828               127.0.0.1:9000
ESTAB          0           0                  192.168.56.100:22              192.168.56.1:24614
ESTAB          0           0                     127.0.0.1:52006               127.0.1.1:8031
ESTAB          0           0                  192.168.56.100:22              192.168.56.1:3063
ESTAB          0           0                     127.0.1.1:8031                127.0.0.1:52006
TIME-WAIT      0           0                     127.0.0.1:37100               127.0.0.1:9000
ESTAB          0           0                     127.0.0.1:9000                127.0.0.1:42828
LISTEN         0           2048                        *:8793                        *:*
LISTEN         0           128                      [::]:22                       [::]:*
(airflow_example) hadoop@mainserver1:~/airflow_example/bin$
```


Sign In - Airflow
192.168.56.100:8080/login/
Airflow
16:24 IST (+05:30)
Log In
Sign In
Enter your login and password below:
Username:
admin
Password:
Sign In


DAGs - Airflow
Executor — Airflow Documents
192.168.56.100:8080/home
Airflow
DAGs
Cluster Activity
Datasets
Security
Browse
Admin
Docs
16:27 IST (+05:30)
FL
Do not use SQLite as metadata DB in production – it should only be used for dev/testing. We recommend using Postgres or MySQL. Click here for more information.
Do not use the SequentialExecutor in production. Click here for more information.
DAGs
All 51
Active 0
Paused 51
Running 0
Failed 0
Filter DAGs by tag
Search DAGs
Auto-refresh
DAG
Owner
Runs
Schedule
Last Run
Next Run
Recent Tasks
dataset_consumes_1
consumes
dataset-scheduled
airflow
Dataset
On s3://dag1/output_1.txt
dataset_consumes_1_and_2
consumes
dataset-scheduled
airflow
Dataset
0 of 2 datasets updated
dataset_consumes_1_never_scheduled
consumes
dataset-scheduled
airflow
Dataset
0 of 2 datasets updated
dataset_consumes_unknown_never_scheduled
dataset-scheduled
airflow
Dataset
0 of 2 datasets updated
dataset_produces_1
dataset-scheduled
produces
airflow
@daily
2024-01-05, 05:30:00
dataset_produces_2
dataset-scheduled
produces
airflow
None
example_bash_operator
example
example2
airflow
0 0 * * *
2024-01-05, 05:30:00


root@mainserver1: ~
(airflow_example) hadoop@mainserver1:~/airflow_example/bin$ airflow config get-value core executor
SequentialExecutor
(airflow_example) hadoop@mainserver1:~/airflow_example/bin$