Advanced SQL, Streaming/Kafka

Week 8

Marilson Campos

UCSC Extension - 2020

Common Table Expressions

Complex queries can get messy.

```
SELECT
   state,
   sum(order_total) AS total_amount
FROM (
      SELECT
       user.user_uid,
       user.user_name,
       user.state as state,
       inv.total AS order_total
   FROM tb_orders inv
   LEFT JOIN tb_user user
      ON  users.user_uid = inv.user_id
   WHERE
      users.user_uid IS NOT NULL
      AND
      inv.order_date = '2020-01-01'
) q1
GROUP BY q1.state
ORDER BY q1.state;
```

Common Table Expressions

Complex queries can get messy – Cleaned version

```
WITH orders_today AS
(
   SELECT
      user.user_uid,
      user.user_name,
      user.state as state,
      inv.total AS order_total
   FROM tb_orders inv
   LEFT JOIN tb_user user
      ON  users.user_uid = inv.user_id
   WHERE
      users.user_uid IS NOT NULL
      AND
      inv.order_date = '2020-01-01'
)

SELECT
   state,
   sum(order_total) AS total_amount
FROM orders_today
GROUP BY state
ORDER BY state;
```

Temporary tables & Hive config variables

Sometimes you want to materialize an intermediary result.

```
-- This is creating a table name variable
SET temp_users=tmp_users_for_day_${hiveconf:dt_date};

DROP TABLE IF EXISTS ${hiveconf:temp_users};

CREATE TEMPORARY TABLE ${hiveconf:temp_users}
AS
  SELECT
     user_id,
     user_name
  FROM
     real_table r_table
  WHERE
     r_ table.dt = '${hiveconf:dt_date}'
  ;
```

Macros

Reuse operations by declaring Hive macros.

```
CREATE TEMPORARY MACRO valid_id(user_id STRING)
(
  CASE WHEN cast(user_id as int) is NULL THEN false ELSE true
  end
);

select valid_id('1a');
false

select valid_id('1');
true

select valid_id(NULL);
false
```

1 – SQL Windows functions

Why we need them?

Answer questions like:

How to get the top 2 salaries of each department?

How to calculate the moving average of the last
3 measurements at each time?

List the cumulative sales for each customer by time?

1 – Anatomy of SQL Windows functions

SELECT
  <WindowFunction> **OVER** <WindowSpec>
FROM <table_name>

WindowFunction =

ROW_NUMBER -> sequence of numbers 1,2,3,4
RANK, DENSE_RANK, -> ranking based on field
NTILE -> specific percentile
SUM -> cumulative sum (requires order by)
COUNT, AVG
LAG -> the value from previous record
LEAD -> the value from next record
FIRST_VALUE -> First record on partition
LAST_VALUE -> Last record on partition

1 – Anatomy of SQL Windows functions

SELECT
  <WindowFunction> **OVER** <WindowSpec>
FROM <table_name>

WindowSpec =
PARTITION BY <field>
ORDER BY <field>
ROWS BETWEEN <boundary> AND <boundary>

<boundary> =
  <x> PRECEDING
  UNBOUNDED PRECEDING
  CURRENT ROW
  <x> FOLLOWING
  UNBOUNDED FOLLOWING

2 – SQL Windows function example

```
The bike_rides table:

  duration INT
  start_time STRING
  end_time STRING
  start_station_number INT
  start_station STRING -- Name of the station
  end_station_number INT
  end_station STRING -- Name of the station
  bike_number STRING
  member_type STRING
```

To load the data: Read file lab_c4_win.txt on the inside of lab_c4_win folder.

2 – Window spec with just ‘order’

```
SELECT
  duration,
  SUM(duration) OVER (ORDER BY start_time) AS running_total
FROM bike_rides
LIMIT 25
;
```

```
1407    1407.0
202     1609.0 = 1407 + 202
519     2128.0 = 1609 + 519
522     2650.0
756     3406.0
551     3957.0
7797    11754.0
```

File: ex_01_with_order.sh

2 – Window spec with ‘order and partition’

```
SELECT
  start_station,
  duration,
  SUM(duration) OVER
    (PARTITION BY start_station ORDER BY start_time) AS running_total
FROM bike_rides;
```

```
10th & E St NW           942     942.0  = 942 + 0
10th & E St NW           447     1389.0 = 447 + 942
10th & E St NW           1054    2443.0
. . .
10th & E St NW           1211    32987.0
10th & Florida Ave NW    472     472.0  = 472 + 0
10th & Florida Ave NW    821     1293.0 = 821 + 472
10th & Florida Ave NW    751     2044.0
10th & Florida Ave NW    1315    3359.0
10th & Florida Ave NW    1059    4418.0
```

File: ex_02_with_order_partition.sh

2 – Window spec with only ‘partition’

```
SELECT
  start_station,
  duration,
  SUM(duration) OVER (PARTITION BY start_station) AS start_station_total
FROM bike_rides;

10th & E St NW           860 32987.0
10th & E St NW           412 32987.0
10th & E St NW           987 32987.0
10th & E St NW           942 32987.0 = sum of values in partition
10th & Florida Ave NW    517 13321.0 = sum of values in partition
10th & Florida Ave NW    672 13321.0
10th & Florida Ave NW    751 13321.0
10th & Florida Ave NW    893 13321.0
```

File: ex_03_with_partition.sh

2 – Window with percentage of partition total

```
SELECT
  start_station,
  duration,
  SUM(duration) OVER (PARTITION BY start_station) AS start_terminal_total,
  round((duration/SUM(duration)
    OVER (PARTITION BY start_station)) * 100, 2) AS pct_of_time
FROM bike_rides
ORDER BY start_station, pct_of_time
;
```

```
10th & E St NW          2191    32987.0 6.64
10th & E St NW          2526    32987.0 7.66
10th & E St NW          3335    32987.0 10.11
10th & E St NW          7084    32987.0 21.48
10th & Florida Ave NW   118     13321.0 0.89
10th & Florida Ave NW   294     13321.0 2.21
10th & Florida Ave NW   419     13321.0 3.15
10th & Florida Ave NW   472     13321.0 3.54
10th & Florida Ave NW   517     13321.0 3.88
```

File: ex_04_2_win.sh

2 – Row number on each start_station

```
SELECT
  start_station,
  start_time,
  duration,
  ROW_NUMBER() OVER
    (PARTITION BY start_station ORDER BY start_time) AS row_number
FROM bike_rides
;
```

```
10th & E St NW          2018-01-28 17:17:56  791     25
10th & E St NW          2018-01-29 15:46:25  630     26
10th & E St NW          2018-01-30 17:34:53  798     27
10th & E St NW          2018-01-31 15:38:12  1211    28
10th & Florida Ave NW   2018-01-01 14:55:34  472     1
10th & Florida Ave NW   2018-01-02 07:13:39  821     2
10th & Florida Ave NW   2018-01-10 07:48:49  751     3
10th & Florida Ave NW   2018-01-10 13:14:26  1315    4
10th & Florida Ave NW   2018-01-12 08:15:30  1059    5
```

File: ex_05_row_num.sh

2 – First 2 rides for each_station

```
SELECT *
FROM (
  SELECT
    start_station, start_time, duration,
    ROW_NUMBER() OVER
      (PARTITION BY start_station ORDER BY start_time) AS row_number
  FROM bike_rides
) q1
WHERE row_number < 3;
```

```
10th & Monroe St NE              2018-01-02 18:16:02  157     1
10th & Monroe St NE              2018-01-07 16:19:17  894     2
10th & U St NW                   2018-01-05 20:19:50  412     1
10th & U St NW                   2018-01-08 23:22:07  290     2
10th St & Constitution Ave NW    2018-01-02 09:49:08  1937    1
10th St & Constitution Ave NW    2018-01-02 15:50:59  761     2
10th St & L'Enfant Plaza SW      2018-01-02 18:11:25  1318    1
10th St & L'Enfant Plaza SW      2018-01-05 17:07:49  1616    2
11th & F St NW                   2018-01-04 07:12:44  196     1
11th & F St NW                   2018-01-07 09:12:32  134     2
```

File: ex_06_top_by_grp.sh

2 – Ranking by minutes

```
SELECT
  start_station,
  duration,
  SUBSTRING(start_time, 1, 13) as time_slot,
  RANK() OVER
   (PARTITION BY start_station ORDER BY SUBSTRING(start_time, 1, 13)) AS
rank
FROM bike_rides;
```

```
10th & E St NW   942 2018-01-03 00    1
10th & E St NW   447 2018-01-06 16    2
10th & E St NW   412 2018-01-08 14    3
10th & E St NW   1054 2018-01-08 14    3
10th & E St NW   503 2018-01-11 00    5 <- there is no rank 4.
10th & E St NW   2191 2018-01-11 15    6
10th & E St NW   2526 2018-01-12 13    7
10th & E St NW   390 2018-01-12 16    8
10th & E St NW   334 2018-01-15 15    9
10th & E St NW   563 2018-01-16 18    10
10th & E St NW   1925 2018-01-17 18    11
```

File: ex_07_rank.sh

2 – Dense ranking by minutes

```
SELECT
  start_station,
  duration,
  SUBSTRING(start_time, 1, 13) as time_slot,
  DENSE_RANK() OVER
   (PARTITION BY start_station ORDER BY SUBSTRING(start_time, 1, 13)) AS
rank
FROM bike_rides;
```

```
10th & E St NW   942 2018-01-03 00    1
10th & E St NW   447 2018-01-06 16    2
10th & E St NW   412 2018-01-08 14    3
10th & E St NW   1054 2018-01-08 14    3
10th & E St NW   503 2018-01-11 00    4 <- Dense does not skip.
10th & E St NW   2191 2018-01-11 15    5
10th & E St NW   2526 2018-01-12 13    6
10th & E St NW   390 2018-01-12 16    7
10th & E St NW   334 2018-01-15 15    8
10th & E St NW   563 2018-01-16 18    9
10th & E St NW   1925 2018-01-17 18    10
10th & E St NW   1068 2018-01-18 14    11
```

File: ex_08_dense_rank.sh

2 – The variation of duration between rides.

```
SELECT
  start_station,
  duration,
  duration - LAG(duration, 1) OVER
    (PARTITION BY start_station ORDER BY duration) AS delta
FROM bike_rides;
```

```
10th & E St NW   1054    NULL <- first row does not have lag.
10th & E St NW   1068    14.0
10th & E St NW   1211    143.0
10th & E St NW   1925    714.0
10th & E St NW   2191    266.0
10th & E St NW   2526    335.0
10th & E St NW   3335    809.0
10th & E St NW   334     -3001.0
10th & E St NW   381     47.0
10th & E St NW   381     0.0
10th & E St NW   390     9.0
10th & E St NW   401     11.0
10th & E St NW   412     11.0
```

File: ex_09_lag.sh

2 – Collecting previous and next record.

```
SELECT
  start_station, duration,
  LAG(duration) OVER window_spec AS prev,
  LEAD(duration) OVER window_spec AS next
FROM bike_rides
ORDER BY start_station, duration
WINDOW window_spec AS (PARTITION BY start_station ORDER BY duration)
```

Station	Curr	Last	Next	
10th & E St NW	***818***	798	823	
10th & E St NW	823	***818***	860	
10th & E St NW	860	823	**942**	
10th & E St NW	**942**	860	983	
10th & E St NW	983	942	987	
10th & E St NW	987	983	NULL	<- No next on last rec
10th & Florida Ave NW	1039	NULL	1059	<- No previous on first rec.
10th & Florida Ave NW	1059	1039	1068	
10th & Florida Ave NW	1068	1059	118	
10th & Florida Ave NW	118	1068	1315	
10th & Florida Ave NW	1315	118	294	
10th & Florida Ave NW	294	1315	419	

File: ex_10_lead_lag.sh

2 – Specifying custom window.

```
SELECT
  start_station, duration,
  SUM(duration) OVER window_spec AS tot
FROM bike_rides
ORDER BY start_station, duration
WINDOW window_spec AS (PARTITION BY start_station ORDER BY duration
    ROWS BETWEEN UNBOUNDED PRECEDING AND 1 PRECEDING);
```

```
10th & E St NW   1054    NULL
10th & E St NW   1068    1054.0 = 1054
10th & E St NW   1211    2122.0 = 1054 + 1068
10th & E St NW   1925    3333.0 = 1054 + 1068 + 1211
10th & E St NW   2191    5258.0
10th & E St NW   2526    7449.0
10th & E St NW   3335    9975.0
10th & E St NW   334     13310.0
10th & E St NW   381     13644.0
10th & E St NW   381     14025.0
10th & E St NW   390     14406.0
```

File: ex_11_win_size.sh

2 – Sum of the 2 previous records.

```
SELECT
  start_station, duration,
  SUM(duration) OVER window_spec AS tot
FROM bike_rides
ORDER BY start_station, duration
WINDOW window_spec AS (PARTITION BY start_station ORDER BY duration
    ROWS BETWEEN 2 PRECEDING AND 1 PRECEDING)

10th & E St NW   1054    NULL
10th & E St NW   1068    1054.0 = 1054
10th & E St NW   1211    2122.0 = 1054 + 1068
10th & E St NW   1925    2279.0.= 1068 + 1211
10th & E St NW   2191    3136.0 = 1211 + 1925
10th & E St NW   2526    4116.0
10th & E St NW   3335    4717.0
10th & E St NW   334     5861.0
10th & E St NW   381     3669.0
10th & E St NW   381     715.0
10th & E St NW   390     762.0
10th & E St NW   401     771.0
```

File: x_12_win_size2.sh

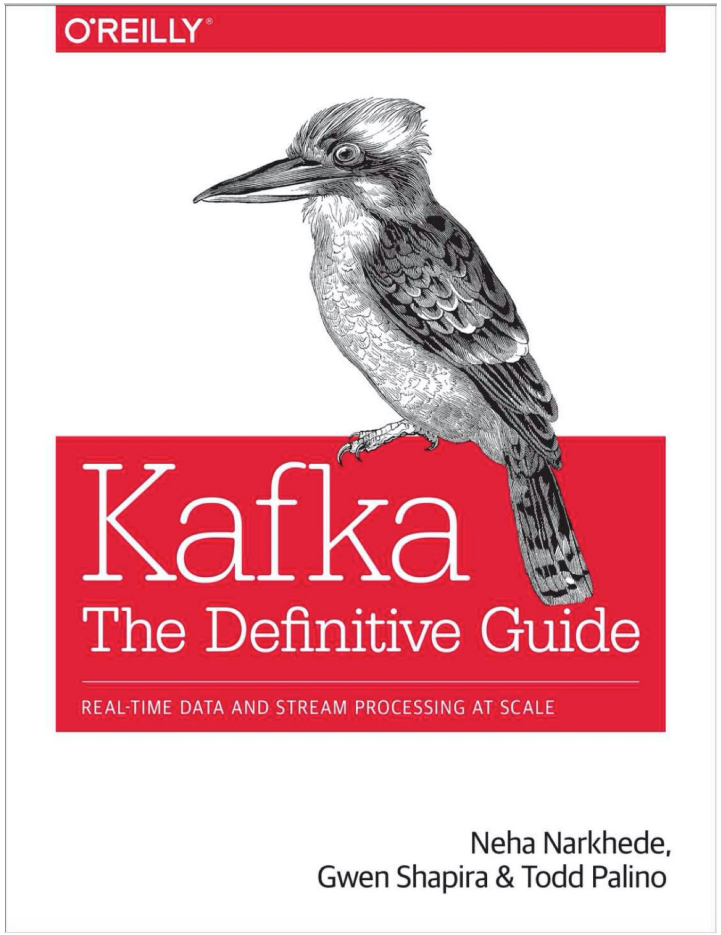
3 – Real time streaming

- Flexible Publish-subscribe/queue

- Robust Replication

- Ordering Preserved at topic partition level

3 – Real time streaming



Recommended reference

- Written by one of the creators of Kafka.
- Currently CTO at Confluent.io

3 – Kafka Marketplace

Apache Kafka – open source

Confluent.io – Apache Kafka as a Service

Amazon MKS – Amazon Apache Kafka as a service

3 – Real time streaming

Use cases

1. Process events in real-time.

2. Most often you need to capture them in real-time
and have the option to analyze some of these events in real-time.

Examples of event sources:

IOT Sensors
Microservice calls
Http calls
etc.

3 – Real time streaming

Realtime Consumers

- Spark Streaming
- Storm
- Apache Flink
- Kinesis
- Kinesis / Analytics

Batch consumers

- Hadoop HDFS
- Aws S3
- HBase
- Cassandra

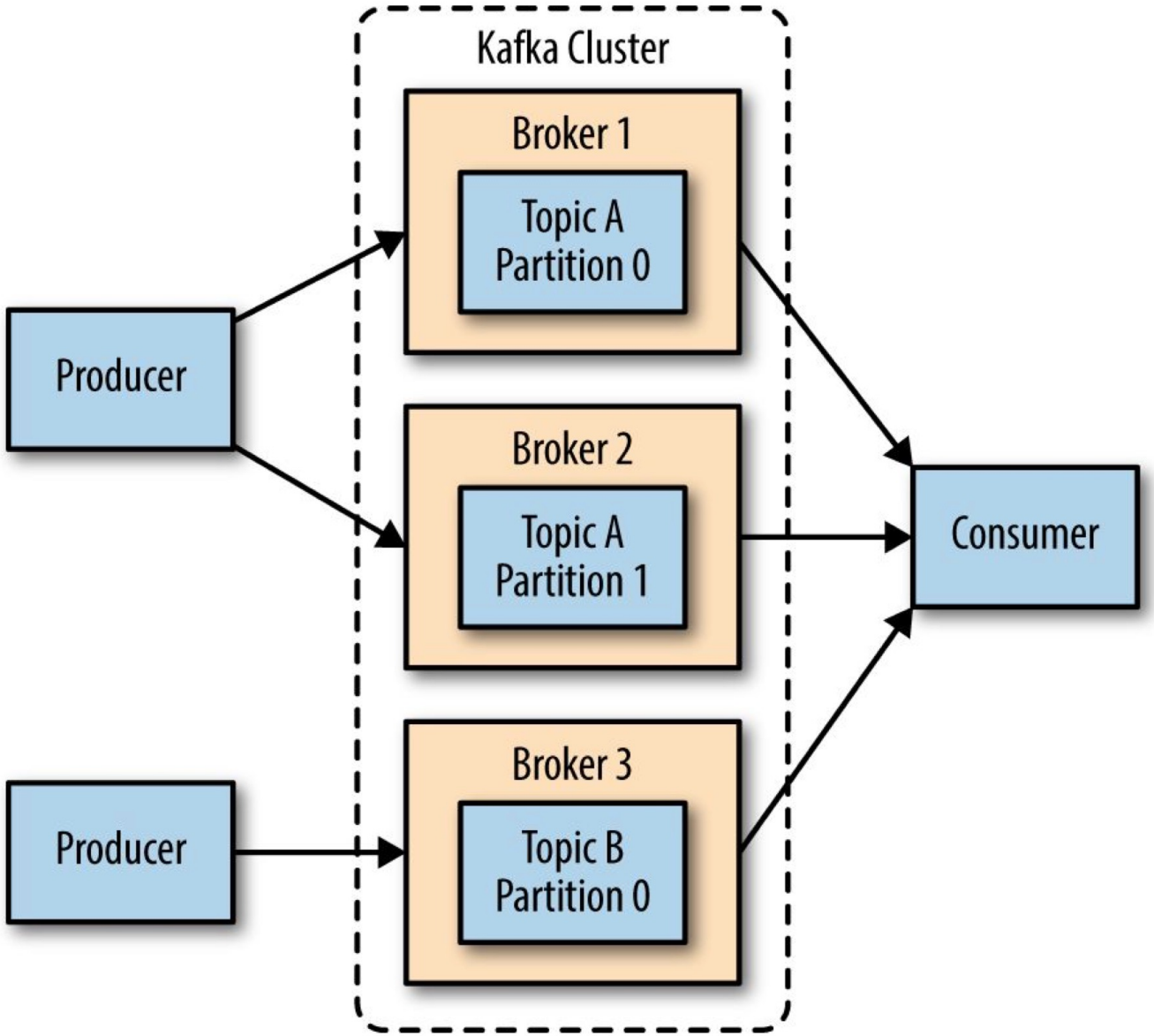
3 – Decoupling from consumer and producers.

- Producers can write faster than consumers for sometime.

- Writes cannot be at higher sustained rate or we run out of space.

- Kafka acts like a reliable system to capture events and deliver some time later to consumers.

3 – Kafka – Cluster components



3 – Kafka - Terminology

Records

have a key (optional), value and timestamp; Immutable

Topic

is a stream of records (“/email_sent”, “/user_signups”)

Topic Log

is the storage for messages for a topic. It consist of Partitions and spread to multiple files on multiple nodes.

Partition

parts of Topic Log

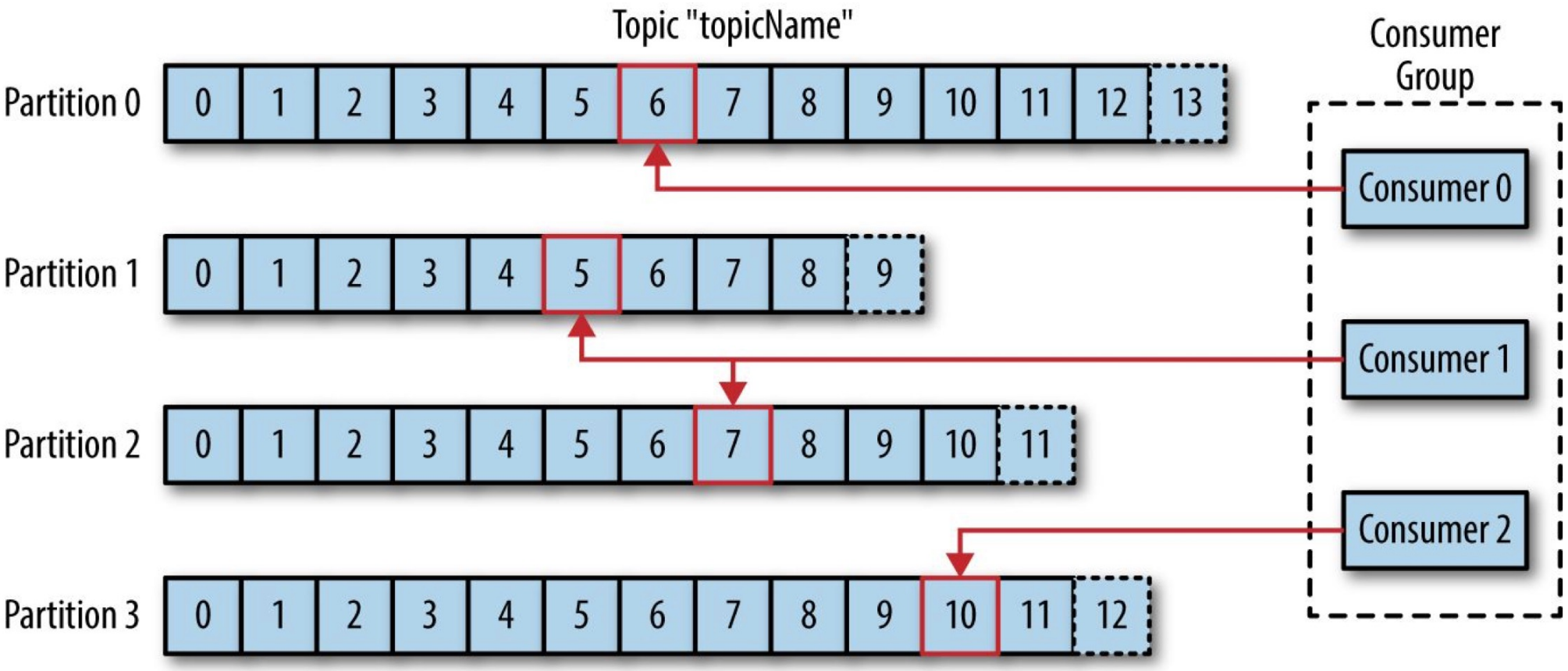
Producer API

is used to produce a streams or records

Consumer API

is used to consume a stream of records

3 – Kafka - Consumption



3 – Kafka - Terminology

Brokers

are Kafka workers and run in a Kafka Cluster.

ZooKeeper

System that coordinates brokers/cluster operation and leadership election

Records

have a key (optional), value and timestamp and are Immutable

Producers

write to Topics

append records at end of Topic log

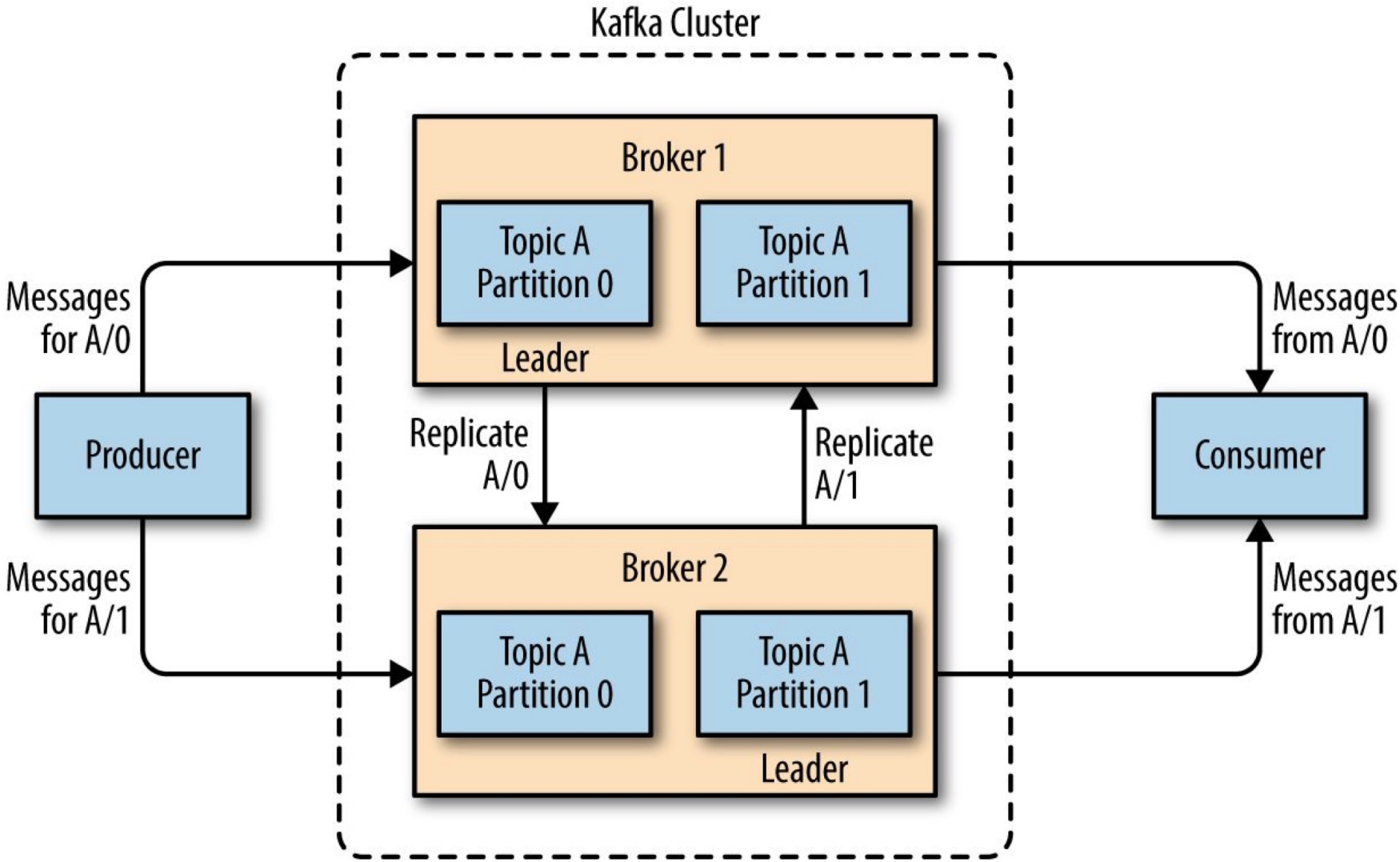
Consumers

read from Topics

can read from Kafka at their own pace

tracks offset from where they left off reading

3 – Kafka Topic Partitions



3 – Kafka Partitions and offsets.

- Order is maintained only in a single partition

- A **partition** is ordered, immutable sequence of records that is continually appended to a commit log.

- **records** in partitions are assigned sequential id number called the **offset**

- Offset identifies each record within the partition

- A **topic partition** must fit on the server that host it

- A **topic** can span many partitions hosted on many servers

3 – Kafka vs Aws Kinesis

- Kinesis Streams is equivalent to Kafka Core

- Kinesis Analytics is equivalent to Kafka Streams

- Kinesis Shard is equivalent to Kafka Partition

- Kinesis Analytics allows you to perform SQL like queries on data streams

- Kafka Streaming allows performing functional aggregations and mutations

- Kafka integrates well with Spark and Apache Flink which allows SQL like queries on streams

4. Cloud Computing types

Infrastructure as a Service (IaaS)

Contains the basic building blocks for cloud IT and typically provide access to networking features, computers and data storage space. IaaS gives you the highest level of flexibility.

Platform as a Service (PaaS)

Removes the need for your organization to manage the underlying infrastructure (usually hardware and operating systems) and allows you to focus on managing of your applications.

Software as a Service (SaaS)

Offers a completed product run and managed by the service provider. You do not have to think about how the service is maintained and only think about how that particular piece of software is going to be used.

4. Cloud Computing types

Infrastructure as a Service (IaaS)

Contains the basic building blocks for cloud IT and typically provide access to networking features, computers and data storage space. IaaS gives you the highest level of flexibility.

Platform as a Service (PaaS)

Removes the need for your organization to manage the underlying infrastructure (usually hardware and operating systems) and allows you to focus on managing of your applications.

Function as a Service & Serverless (FaaS)

Offers capability to call functions in the cloud. You do not have to think about how the service is maintained, scaled or recovered.

Computing Deployment models

- Cloud

- Hybrid

- In Premises

Computing Deployment models

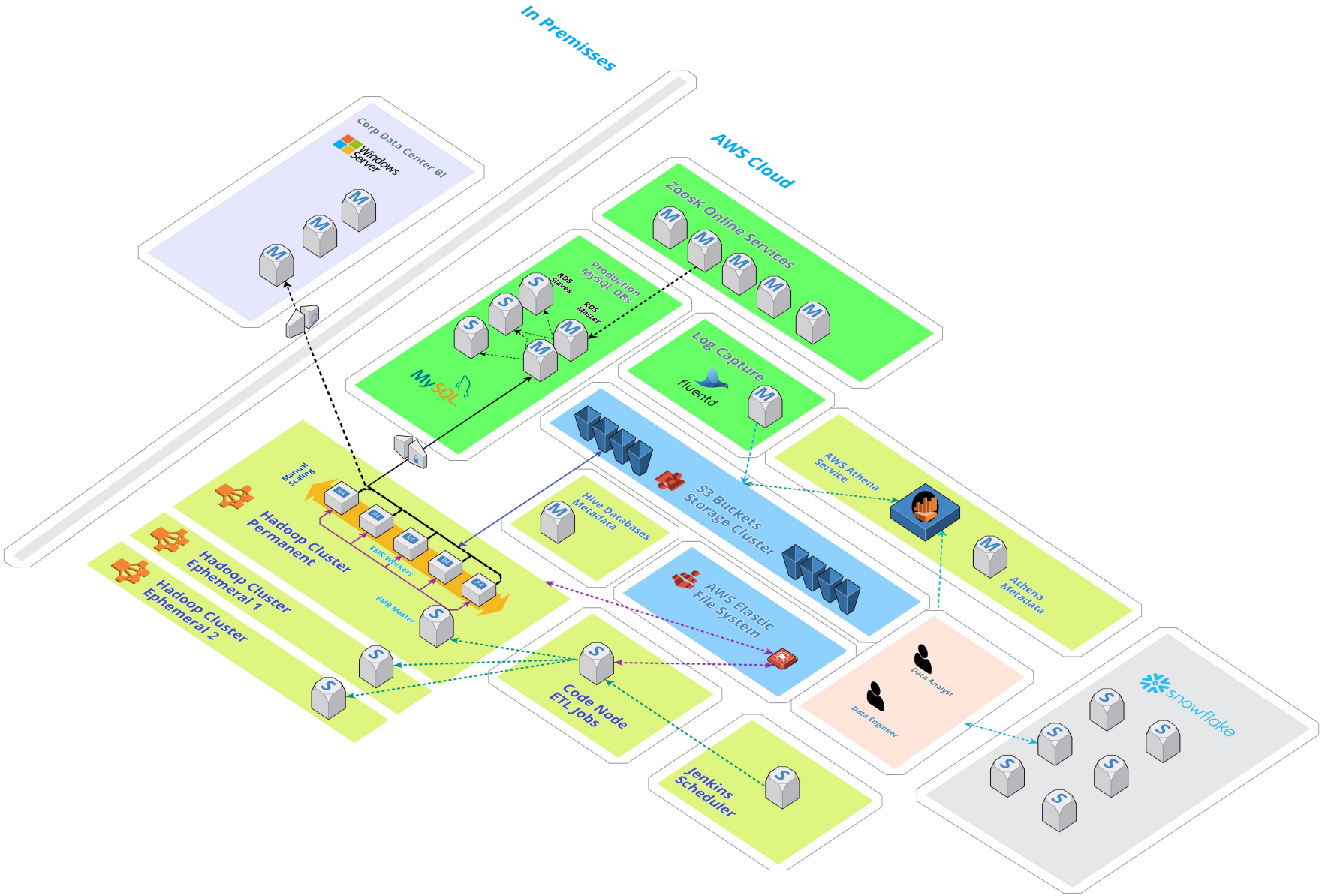
Cloud

Applications are fully-deployed on the cloud, with all components running on it.

Applications on the cloud have either been created for the cloud or have been migrated from data-center to take advantage of the benefits of cloud computing.

They can be built on low-level services or use fully managed services. (more about this later…)

Example Data Architecture - Cloud



Computing Deployment models

Hybrid

The deployment of resources on-premises, using virtualization and resource management tools, is sometimes called the “private cloud.”

In-premises deployment doesn’t provide many of the benefits of cloud computing, but it can enable deployment of dedicated resources.

In most cases, this deployment model is the same as legacy IT infrastructure while using application management and virtualization technologies to optimize resource utilization.

Computing Deployment models

In Premises

It’s been the traditional way of building systems where company uses a datacenter and is responsible to buy, deploy and manage all the hardware.

This approach is very limited during re-architecting efforts. Also is expensive as companies need to plan to handle peaks.

Amazon EC2

Amazon Elastic Compute Cloud is a service that allows dynamically allocating “instances” of servers inside of the Amazon cloud.

There are hundreds of types of hosts with the most different hardware resources.

Once an instance is started your account is charged by the number of hours online.

For more details, take a look at:

https://www.ec2instances.info/

Amazon S3

Amazon Simple Storage Service (s3) is a service that allows storing massive amounts of data into a fully redundant infrastructure.

Data is represented as files and its possible to ‘simulate’ a hierarchical directory tree.

Data is globally available and secure.

There are several levels of performance and pricing on S3.

For more details take a look at:
https://aws.amazon.com/s3/

Amazon EMR

Amazon Elastic Map/Reduce is a service that allows creating Hadoop clusters with flexibility of components used and cluster capacity.

Clusters can be launched and kept running for many days.

Cluster can be created to execute a daily task in the morning, compute the task and shut down.

Pay for the minutes the cluster nodes are running.

Note: There is a small additional cost creating and shutting down a cluster immediately.

Cloud Computing: Main Advantages

- Facilitates SAAS (Software as a Service)
  - Simple deployment, simulation and upgrades.
- Flexible capacity allocation.
  - No need to plan for peaks.
- Always available
  - Redundancy is embedded from ground up.
- Agile during Re-Architectural efforts
  - No need to commit to any hardware.
  - Easy to try new designs and measure cost benefit.
- Enables Adding Managed Services offerings on top of basic services.
  - Amazon offers managed databases services (RDS) and managed Hadoop cluster services (EMR).

AWS – There are many services. Focus!

Amazon Web Services

Compute

- **EC2** – Virtual Servers in the Cloud
- **Lambda** – Run Code in Response to Events
- **EC2 Container Service** – Run and Manage Docker Containers

Storage & Content Delivery

- **S3** – Scalable Storage in the Cloud
- **Elastic File System** PREVIEW – Fully Managed File System for EC2
- **Storage Gateway** – Integrates On-Premises IT Environments with Cloud Storage
- **Glacier** – Archive Storage in the Cloud
- **CloudFront** – Global Content Delivery Network

Database

- **RDS** – MySQL, Postgres, Oracle, SQL Server, and Amazon Aurora
- **DynamoDB** – Predictable and Scalable NoSQL Data Store
- **ElastiCache** – In-Memory Cache
- **Redshift** – Managed Petabyte-Scale Data Warehouse Service

Networking

- **VPC** – Isolated Cloud Resources
- **Direct Connect** – Dedicated Network Connection to AWS
- **Route 53** – Scalable DNS and Domain Name Registration

Administration & Security

- **Directory Service** – Managed Directories in the Cloud
- **Identity & Access Management** – Access Control and Key Management
- **Trusted Advisor** – AWS Cloud Optimization Expert
- **CloudTrail** – User Activity and Change Tracking
- **Config** – Resource Configurations and Inventory
- **CloudWatch** – Resource and Application Monitoring
- **Service Catalog** – Personalized Catalog of AWS Resources

Deployment & Management

- **Elastic Beanstalk** – AWS Application Container
- **OpsWorks** – DevOps Application Management Service
- **CloudFormation** – Templated AWS Resource Creation
- **CodeDeploy** – Automated Deployments
- **CodeCommit** – Managed Git Repositories
- **CodePipeline** – Continuous Delivery

Analytics

- **EMR** – Managed Hadoop Framework
- **Kinesis** – Real-time Processing of Streaming Big Data
- **Data Pipeline** – Orchestration for Data-Driven Workflows
- **Machine Learning** – Build Smart Applications Quickly and Easily

Application Services

- **SQS** – Message Queue Service
- **SWF** – Workflow Service for Coordinating Application Components
- **AppStream** – Low Latency Application Streaming
- **Elastic Transcoder** – Easy-to-use Scalable Media Transcoding
- **SES** – Email Sending Service
- **CloudSearch** – Managed Search Service
- **API Gateway** – Build, Deploy and Manage APIs

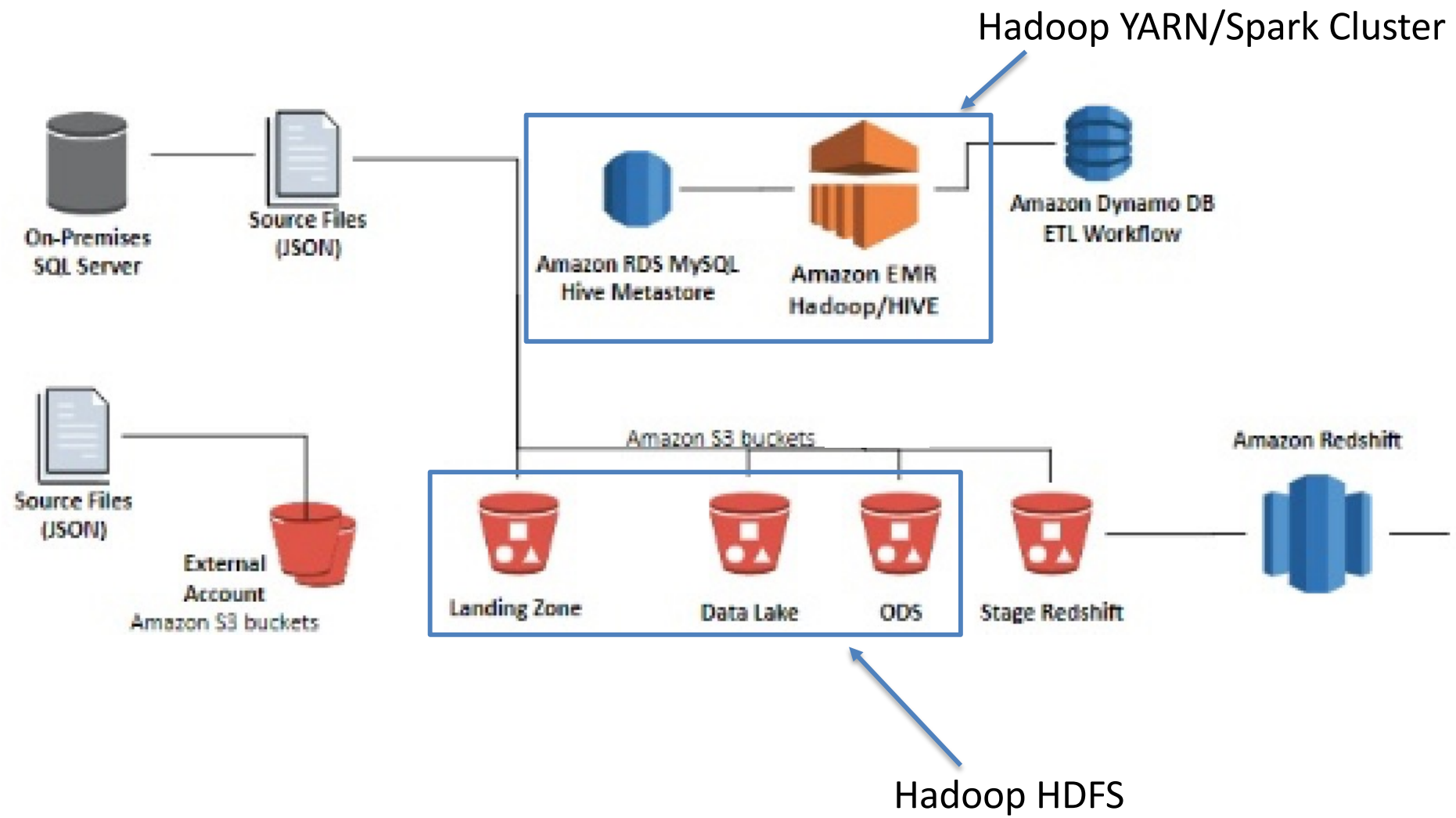
Mobile Services

- **Cognito** – User Identity and App Data Synchronization
- **Device Farm** – Test Android, Fire OS, and iOS apps on real devices in the Cloud
- **Mobile Analytics** – Collect, View and Export App Analytics
- **SNS** – Push Notification Service

Enterprise Applications

- **WorkSpaces** – Desktops in the Cloud
- **WorkDocs** – Secure Enterprise Storage and Sharing Service
- **WorkMail** PREVIEW – Secure Email and Calendaring Service

AWS for Big Data: EC2, EMR, S3 and DynamoDB



4. Apache Airflow

Airflow - Terminology

DAG

Directed acyclic graph

Tasks

a DAG is composed of tasks

Operator

Building block to create tasks.

Task Instance

An specific parametrized run of task

Hooks

provide connectivity to external systems.

Connections

Keep the credentials independent of code.

Airflow - Terminology

Queues

Allow to allocate capacity

XComs

a channel to write and read data while running a DAG

Variable

Building block to create tasks.

Airflow - Operators

BashOperator - executes a bash command

PythonOperator - calls an arbitrary Python function

EmailOperator - sends an email

SimpleHttpOperator - sends an HTTP request

MySqlOperator, PostgresOperator,
MsSqlOperator, OracleOperator
- Connects to each of the databases.

JdbcOperator – executes SQK via jdbc connection

Airflow - Operators

DockerOperator – Interacts with docker containers

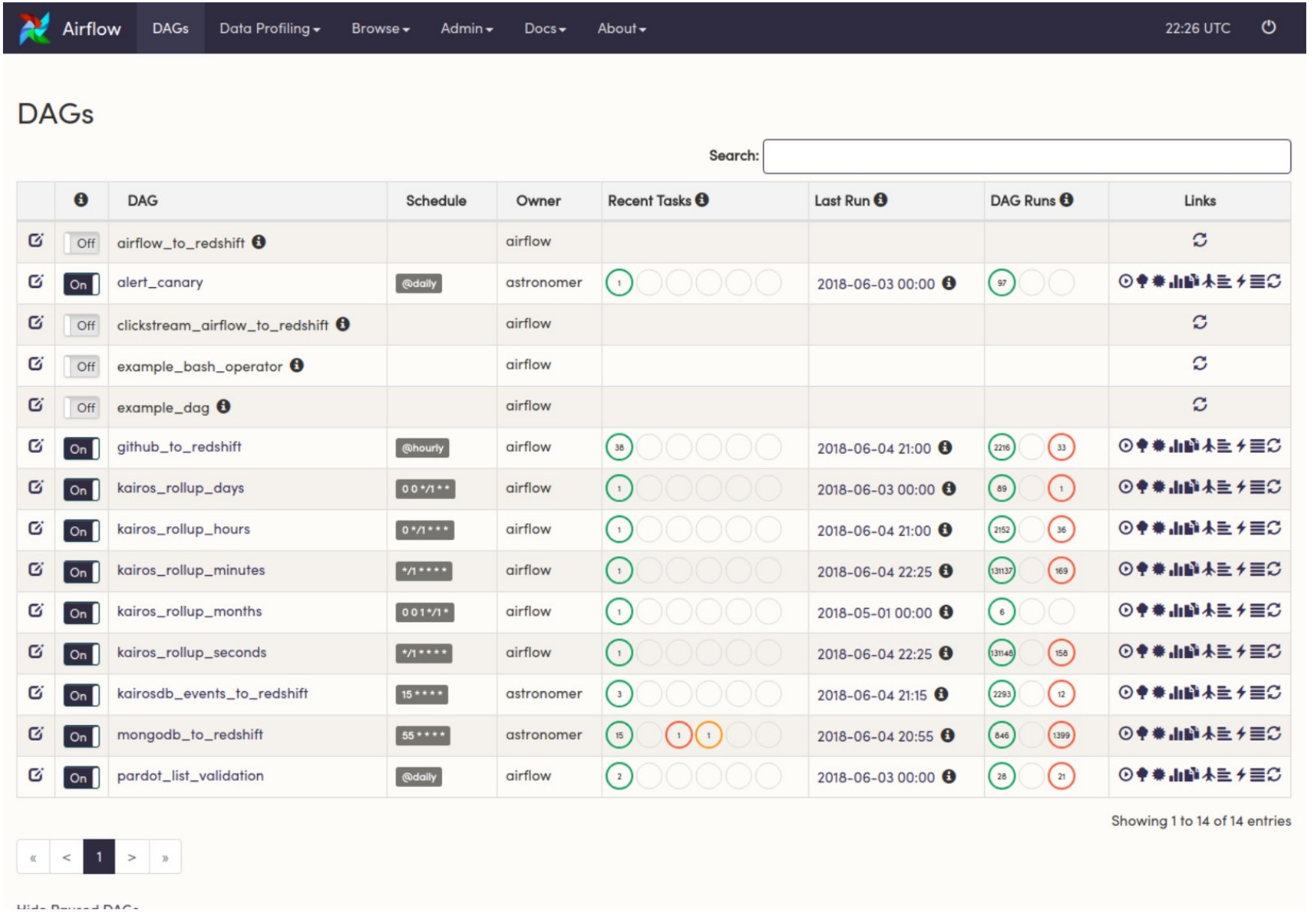
HiveOperator – execute hive queries

S3FileTransformOperator – Store files in S3

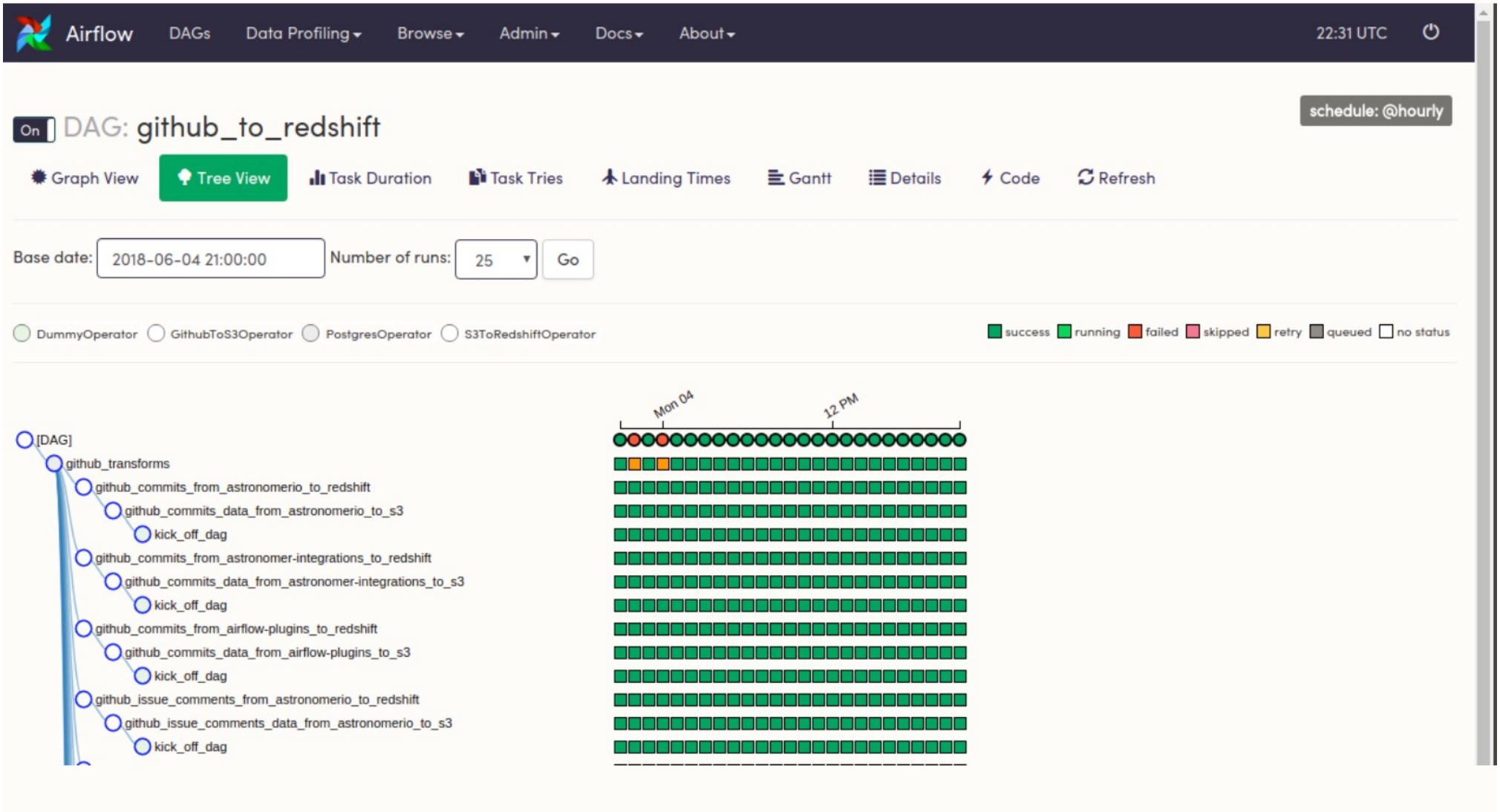
PrestoToMySqlTransfer – Transfer files from presto to MySQL

SlackAPIOperator – Send slack messages

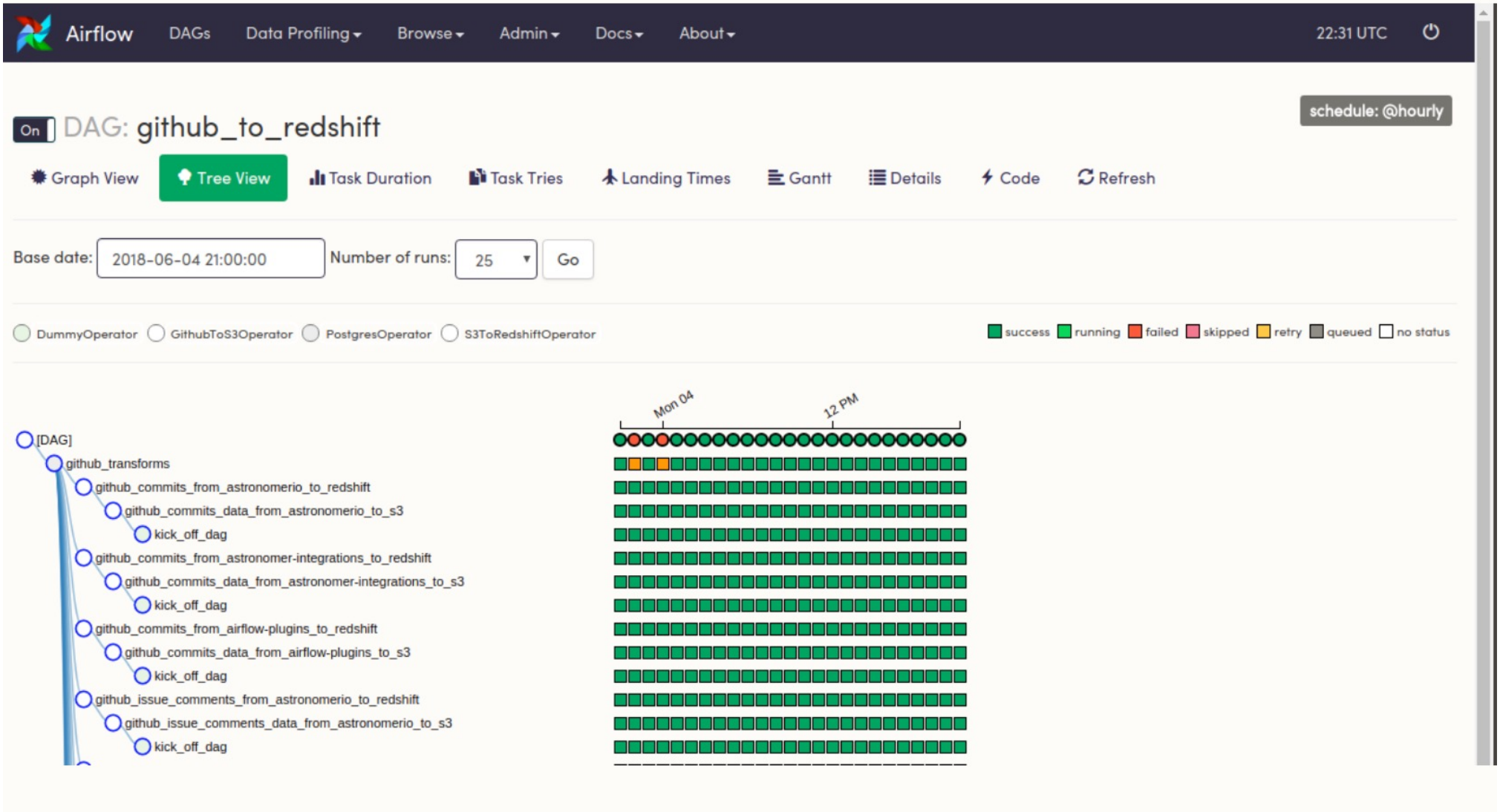
Airflow – Dags View



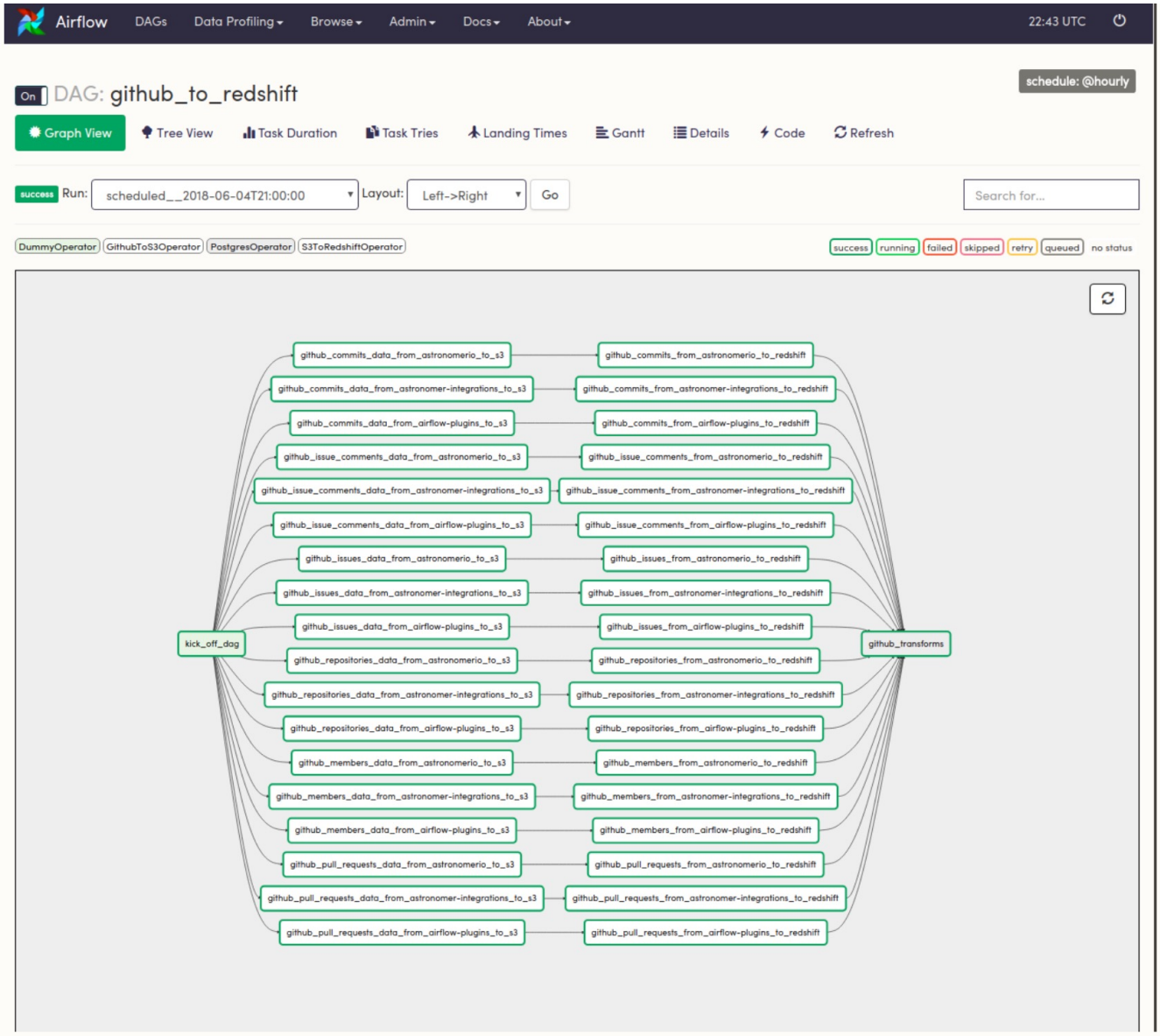
Airflow – Dag tree view



Airflow – Dag tree view



Airflow – Dag graph view



Airflow – Sample code

```
dag = DAG('my_dag', start_date=datetime(2019, 1, 1))

explicit_op = DummyOperator(task_id='op1', dag=dag)

deferred_op = DummyOperator(task_id='op2')
deferred_op.dag = dag

# equivalent operations
op1 >> op2
op1.set_downstream(op2)

# equivalent operations
op2 << op1
op2.set_upstream(op1)
```

ETL Design – Step to create a pipeline.

Step 1. **Figure out the tables and schemas you need.**

Step 2. **Create each step in bash and test the manual process first.**

Step 3. **Integrate the commands into a orchertration tool (airflow or custom program).**

Step 4. **Perform end-to-end test using the deployed ETL.**