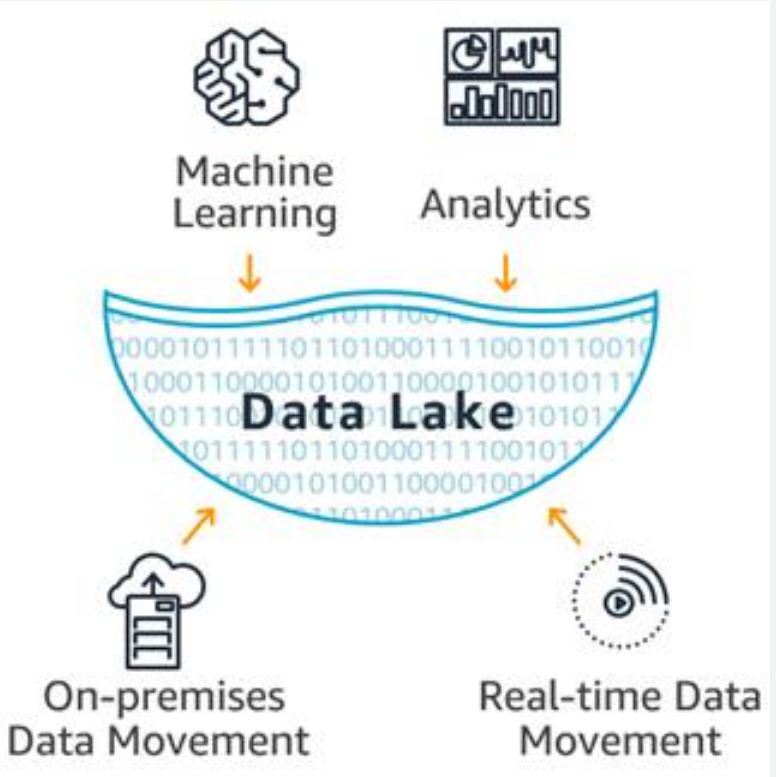
# Building a Data Lake on AWS

Paige Broderick

Solutions Architect
Amazon Web Services

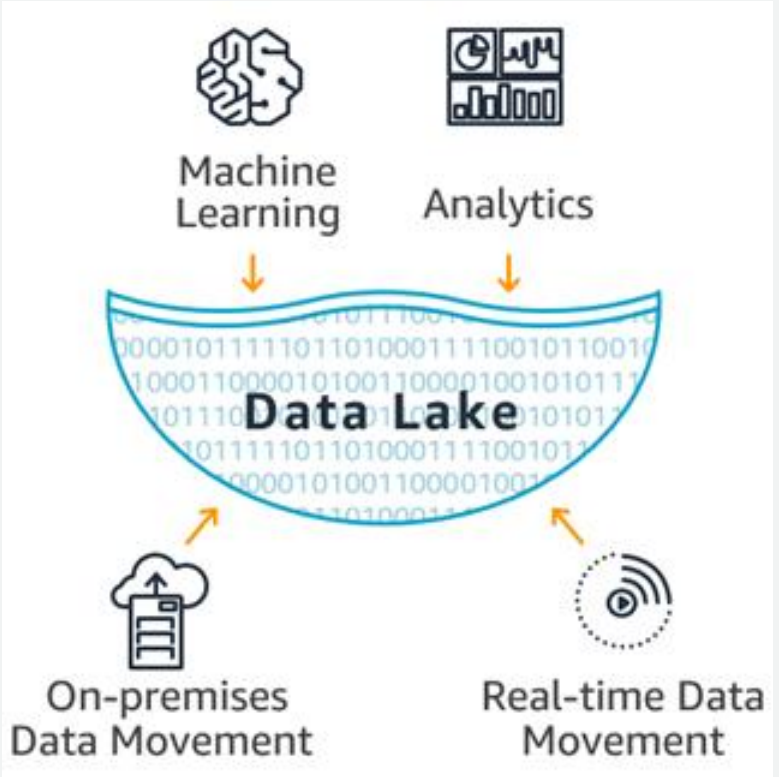
# What is a Data Lake?

- A centralized repository for both structured and unstructured data
- Store data as-is in open-source file formats to enable direct analytics

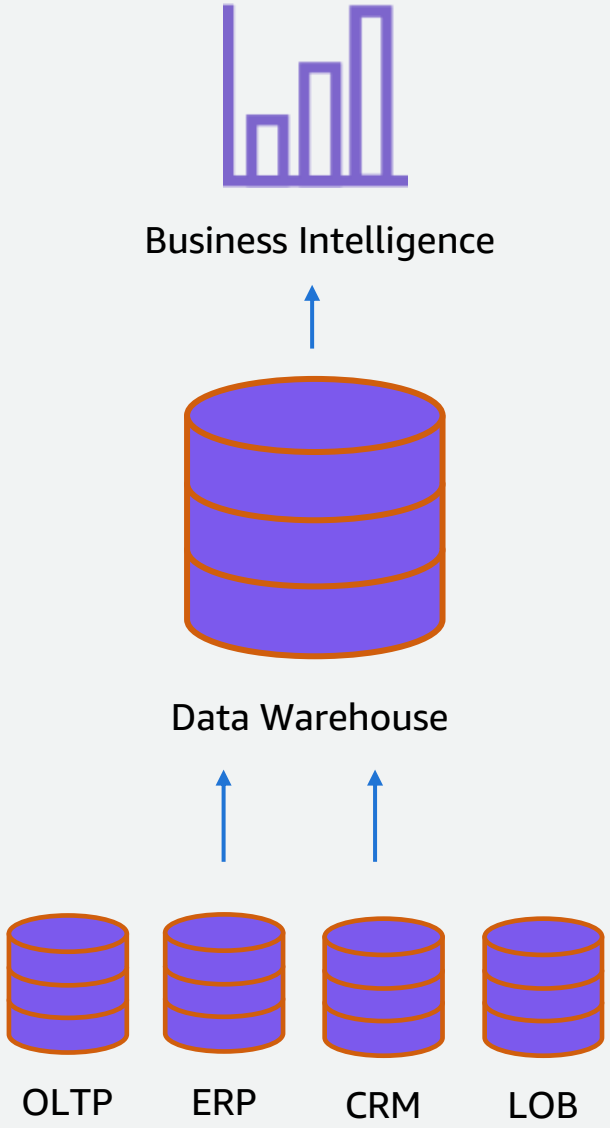


# Why a Data Lake?

- Decouple storage from compute, allowing you to scale
- Enable advanced analytics across all of your data sources
- Reduce complexity in ETL and operational overhead
- Future extensibility as new database and analytics technologies are invented



# Traditionally, Analytics Looked Like This




Relational Data

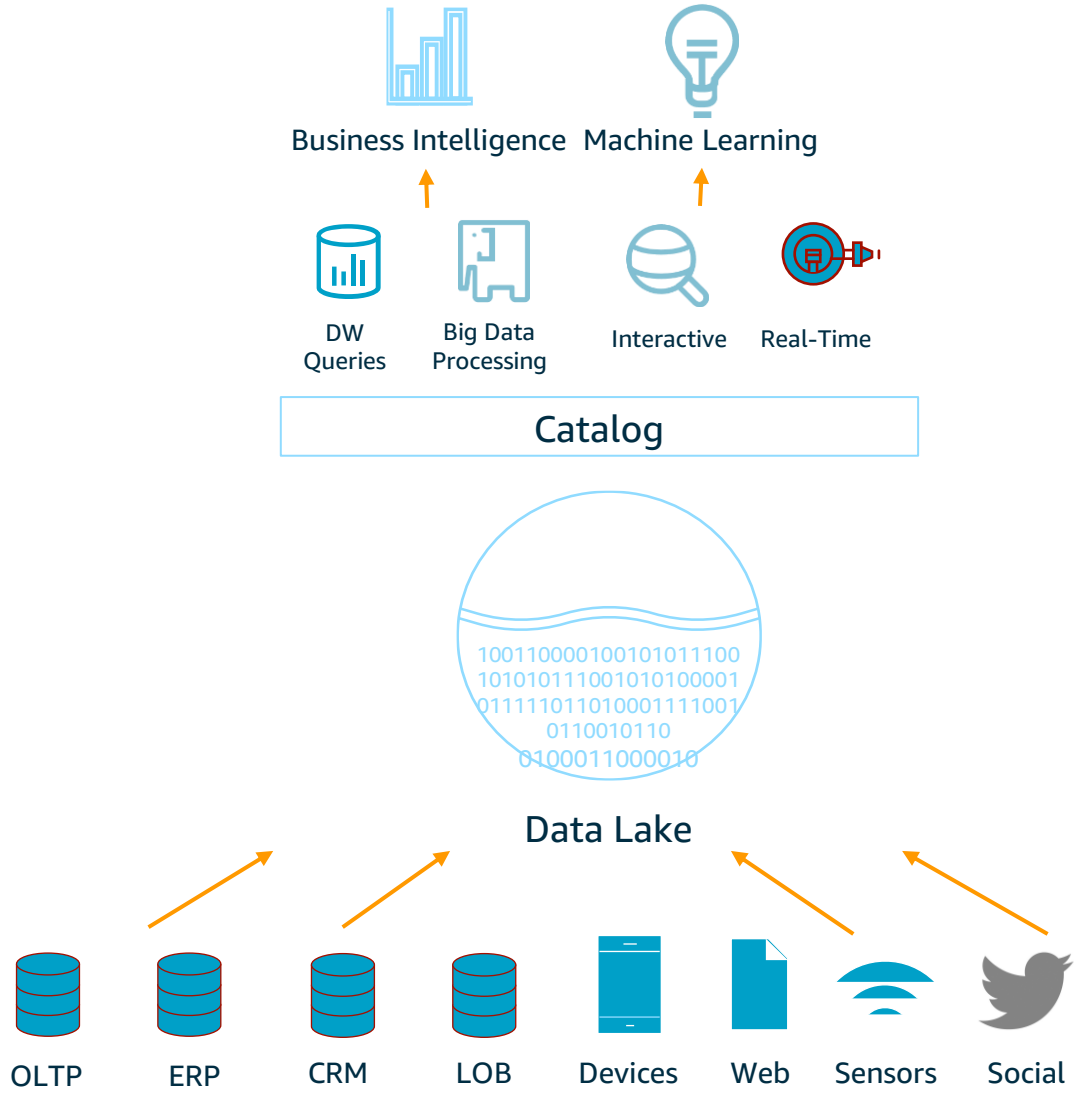
TBs-PBs Scale

Schema Defined Prior to Data Load

Operational and Ad Hoc Reporting

Large Initial Capex + $$K / TB/ Year

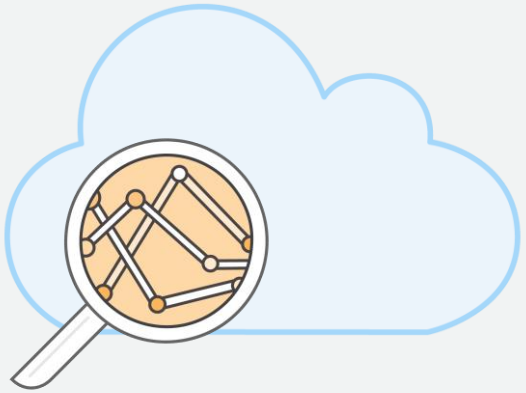
# Data Lakes Extend the Traditional Approach




- TB-EBs Scale
- All Data in one place, a Single Source of Truth
- Relational and Non-Relational Data
- Decouples (low cost) Storage and Compute
- Schema on Read
- Diverse Analytical Engines

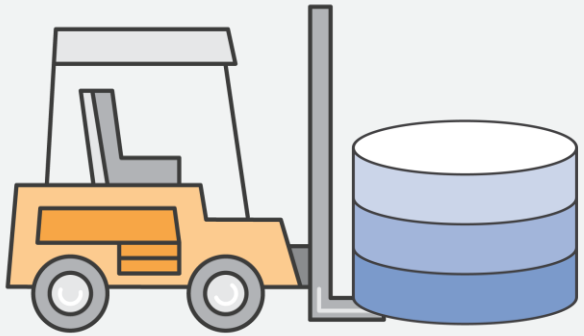
# Benefits of a Data Lake – All Data in One Place

"Why is the data distributed in many locations? Where is the single source of truth ?"

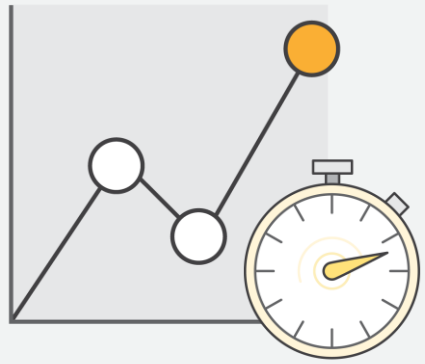


Store and analyze all of your data, from all of your sources, in one centralized location.

# Benefits of a Data Lake – Quick Ingest

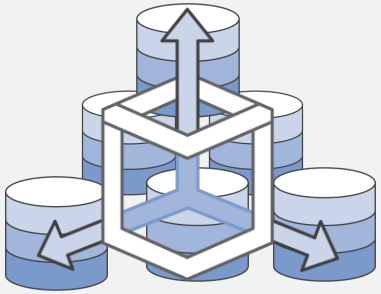


"How can I collect data quickly from various sources and store it efficiently?"

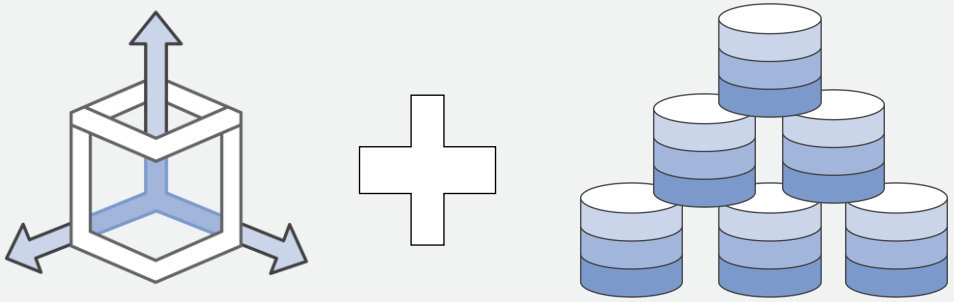


Quickly ingest data without needing to force it into a pre-defined schema.

# Benefits of a Data Lake – Storage vs Compute

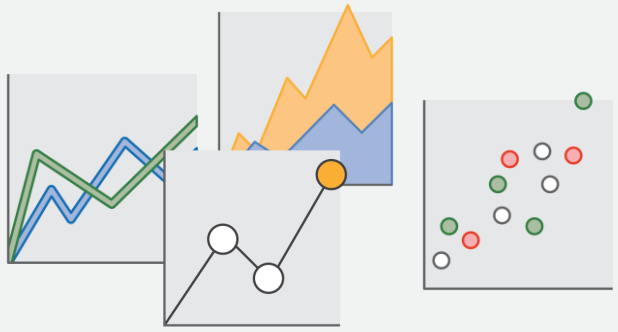


"How can I scale up with the volume of data being generated?"



Separating your storage and compute allows you to scale each component as required

# Benefits of a Data Lake – Schema on Read



"Is there a way I can apply multiple analytics and processing frameworks to the same data?"

A Data Lake enables ad-hoc analysis by applying schemas on read, not write.

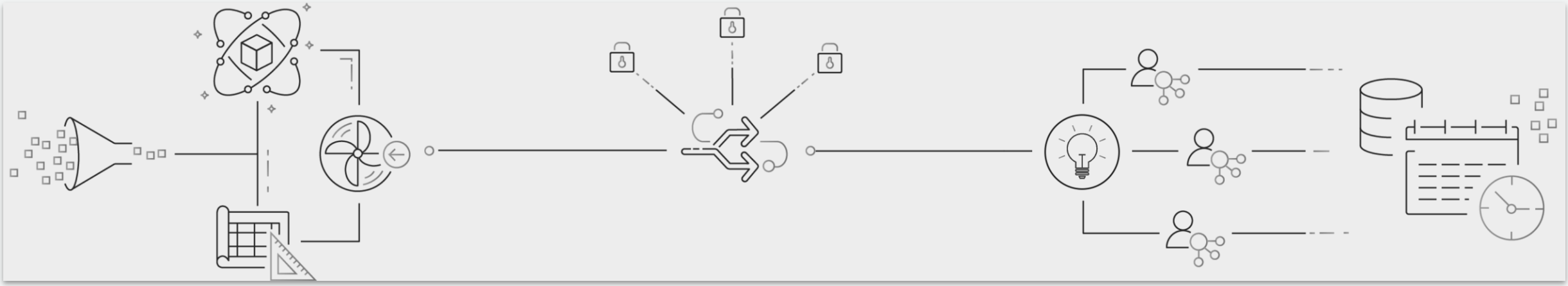
# Building a Data Lake on AWS

# Why AWS?

Implementing a Data Lake architecture requires a broad set of tools and technologies to serve an increasingly diverse set of applications and use cases.

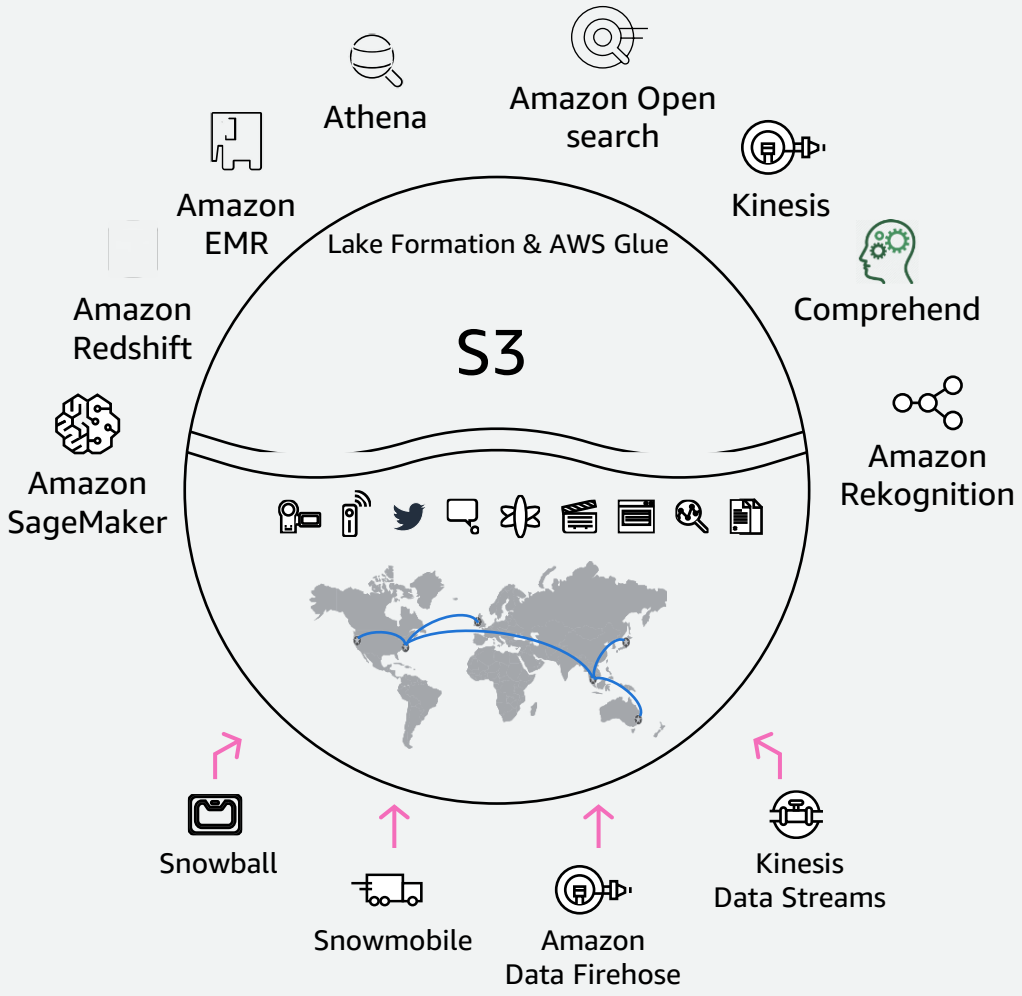
## Typical steps for building a data lake

1. Set up storage



2. Move data
3. Cleanse, prep, and catalog data
4. Configure and enforce security and compliance policies
5. Make data available for analytics

# Robust data lake infrastructure with AWS




Durable and available; exabyte scale

Secure, compliant, auditable

Object-level controls for fine-grained access

Fast performance by retrieving subsets of data

Decoupling of compute and storage

On-demand resources, tiering, cost choices

# Why Amazon S3 for a Data Lake?

## Durable

Designed for **11 9s** of durability

## Available

Designed for **99.99**% availability

## High performance

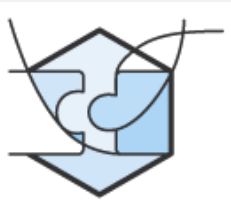
- Multiple upload
- Range GET

## Easy to use

- Simple REST API
- AWS SDKs
- Read-after-create consistency
- Event notification
- Lifecycle policies

## Scalable

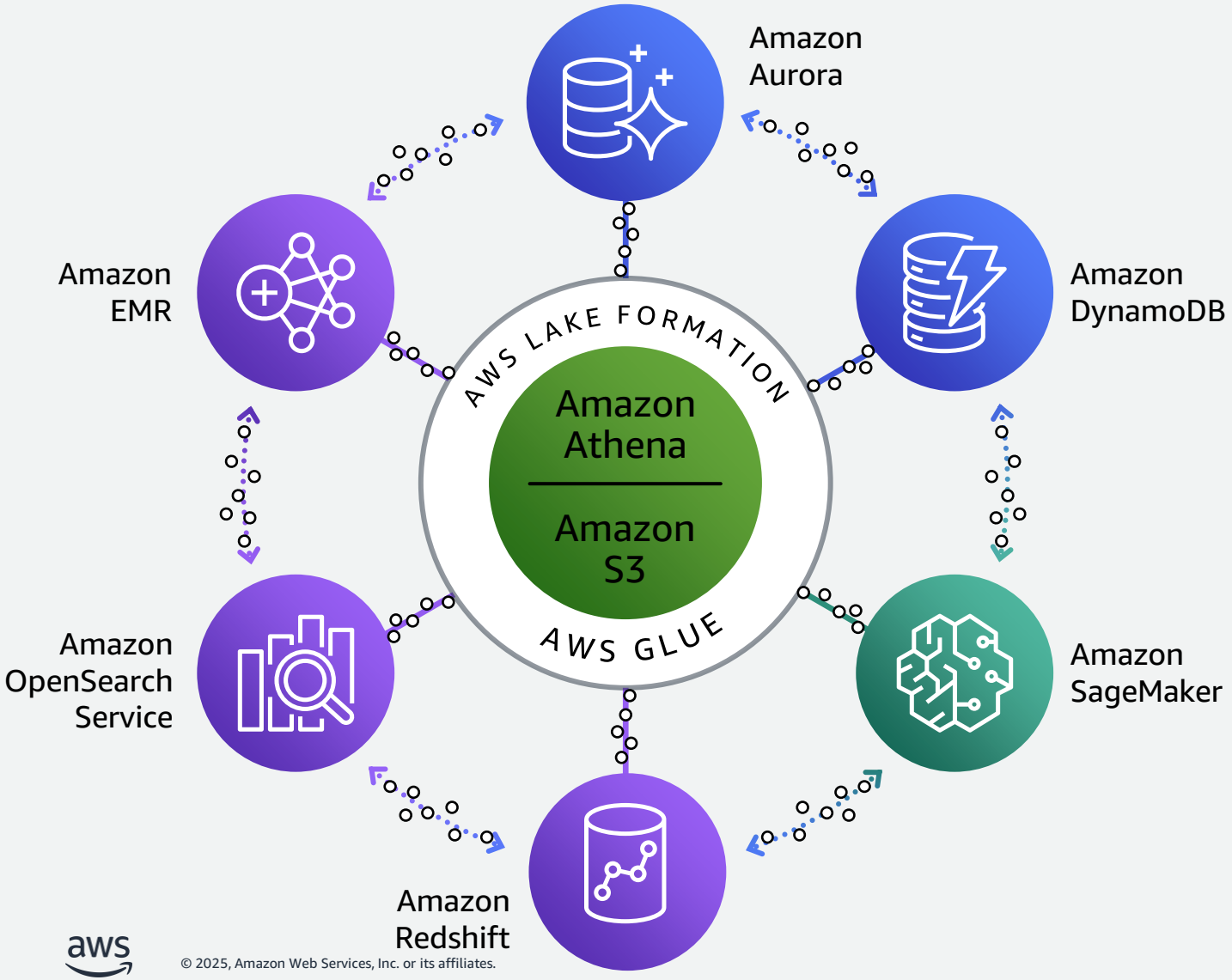
- Store as much as you need
- Scale storage and compute independently
- No minimum usage commitments



## Integrated

- Amazon EMR
- Amazon Redshift
- Amazon DynamoDB
- Amazon SageMaker
- Many more

# Lake House architecture on AWS




SCALABLE DATA LAKES

PURPOSE-BUILT DATA SERVICES

SEAMLESS DATA MOVEMENT

UNIFIED GOVERNANCE

PERFORMANT AND COST-EFFECTIVE

# AWS Lake Formation


- Build a secure data lake in days

## Build data lakes quickly

Move, store, and catalog your data faster

## Simplify security management

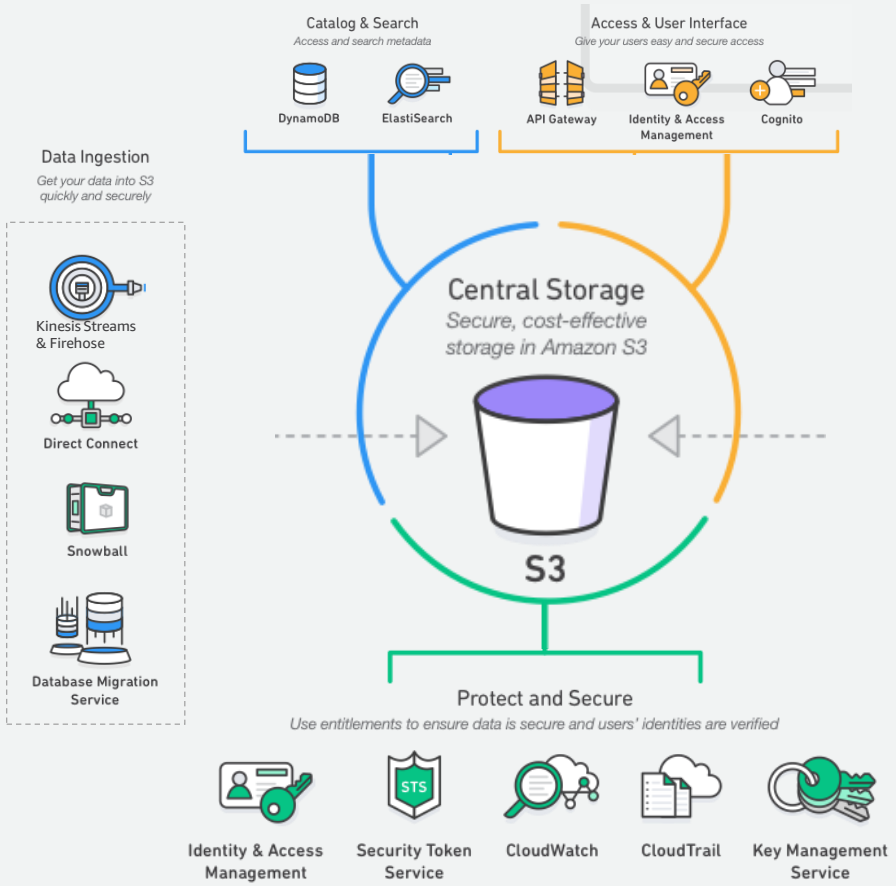
Centrally define security, governance, and auditing policies in one place

## Provide self-service access to data

Build a catalog of datasets

Define user access policies

# Future proof and Integrated DataLake




## Processing & Analytics

### Real-time

- OpenSearch Service
- Spark Streaming on EMR
- AWS Lambda
- Managed service for Flink, Kinesis Data Streams
- Apache Flink on EMR
- Apache Storm on EMR

### Analytics

- EMR Hadoop, Spark, Presto
- Athena Query Service
- Redshift Data Warehouse

### AI & Predictive

- Amazon Lex Speech recognition
- Amazon Polly Text to speech
- Amazon Rekognition
- Machine Learning Predictive analytics
- SageMaker

### Transactional & RDBMS

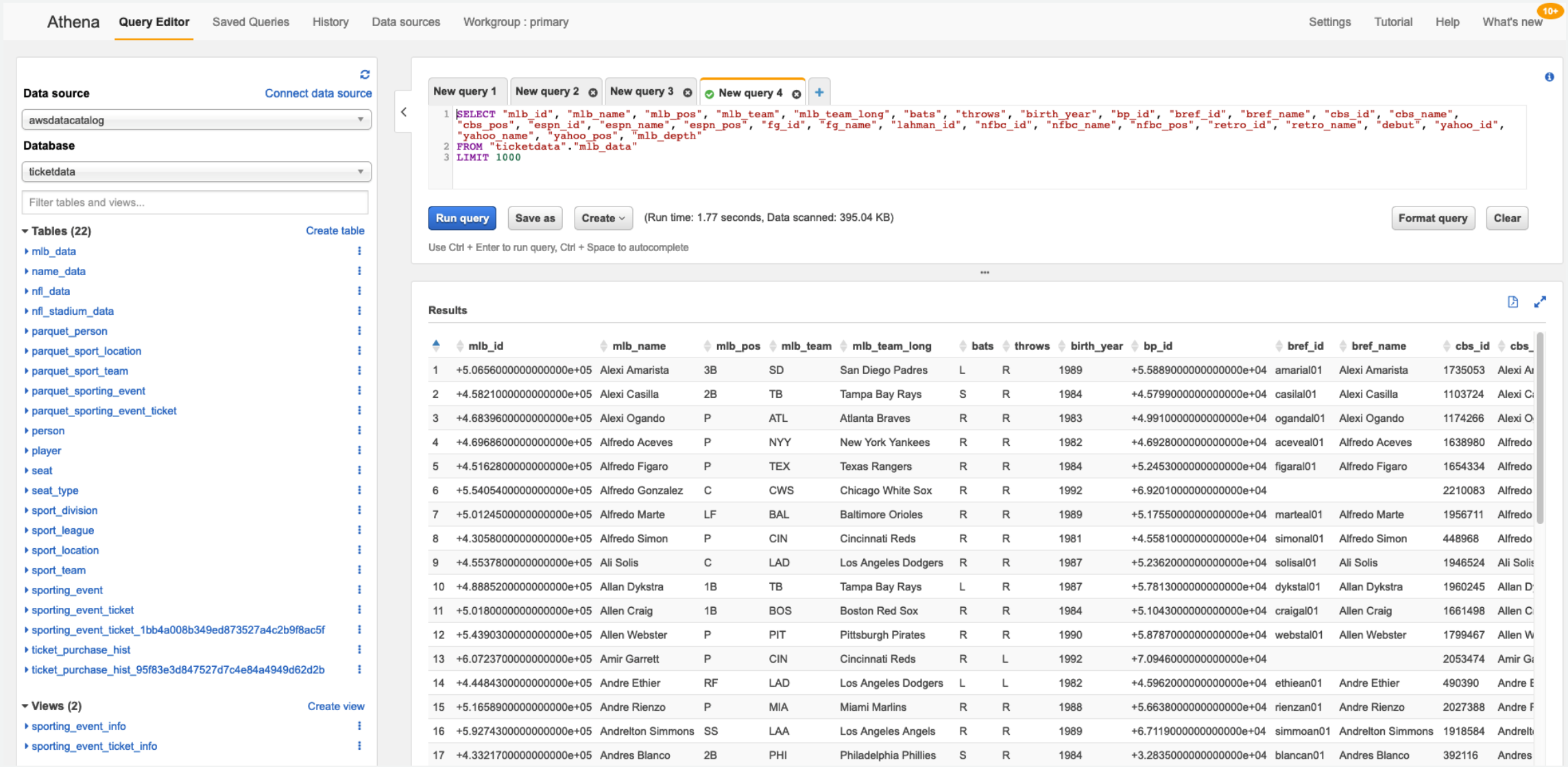
- DynamoDB NoSQL DB
- Aurora Relational Database

### BI & Data Visualization

- Amazon QuickSight
- tableau
- MicroStrategy
- kibana

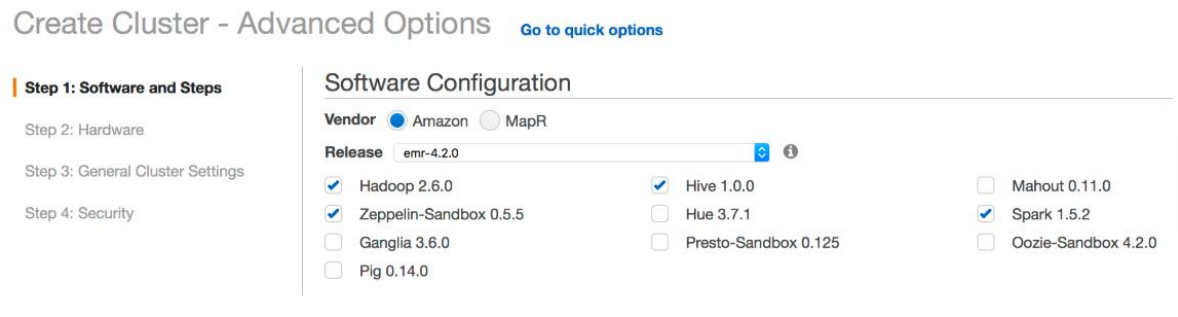
# What can you do with a Data Lake?

# Query Directly with Amazon Athena

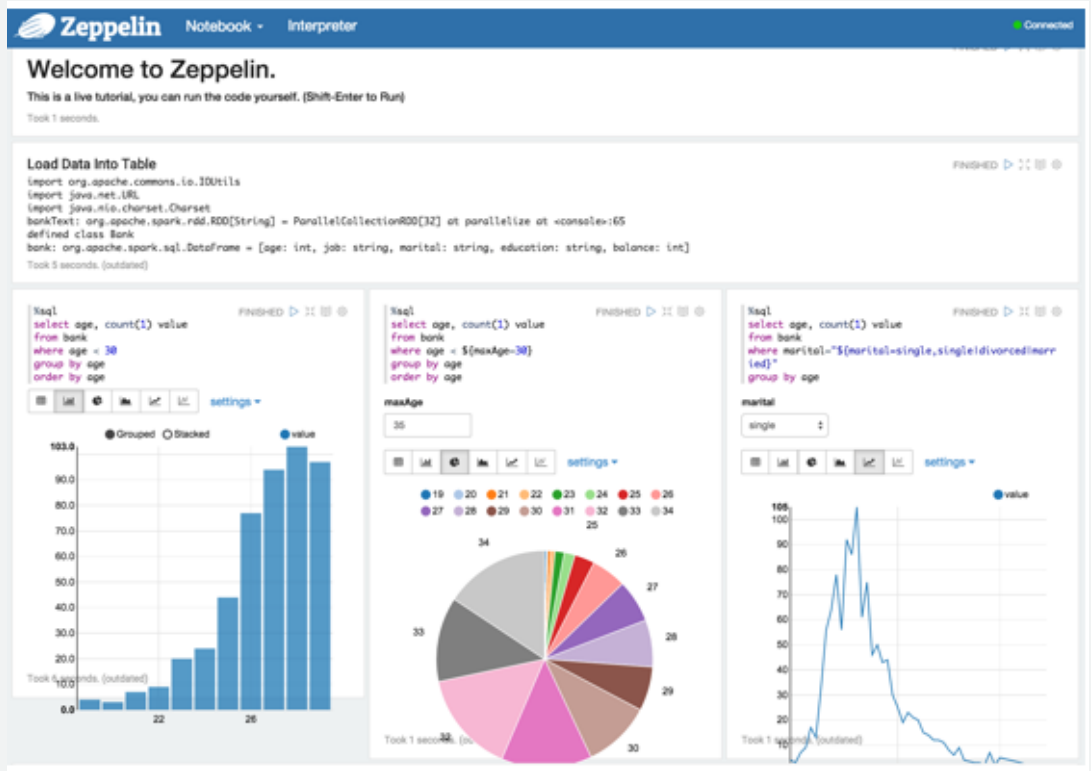


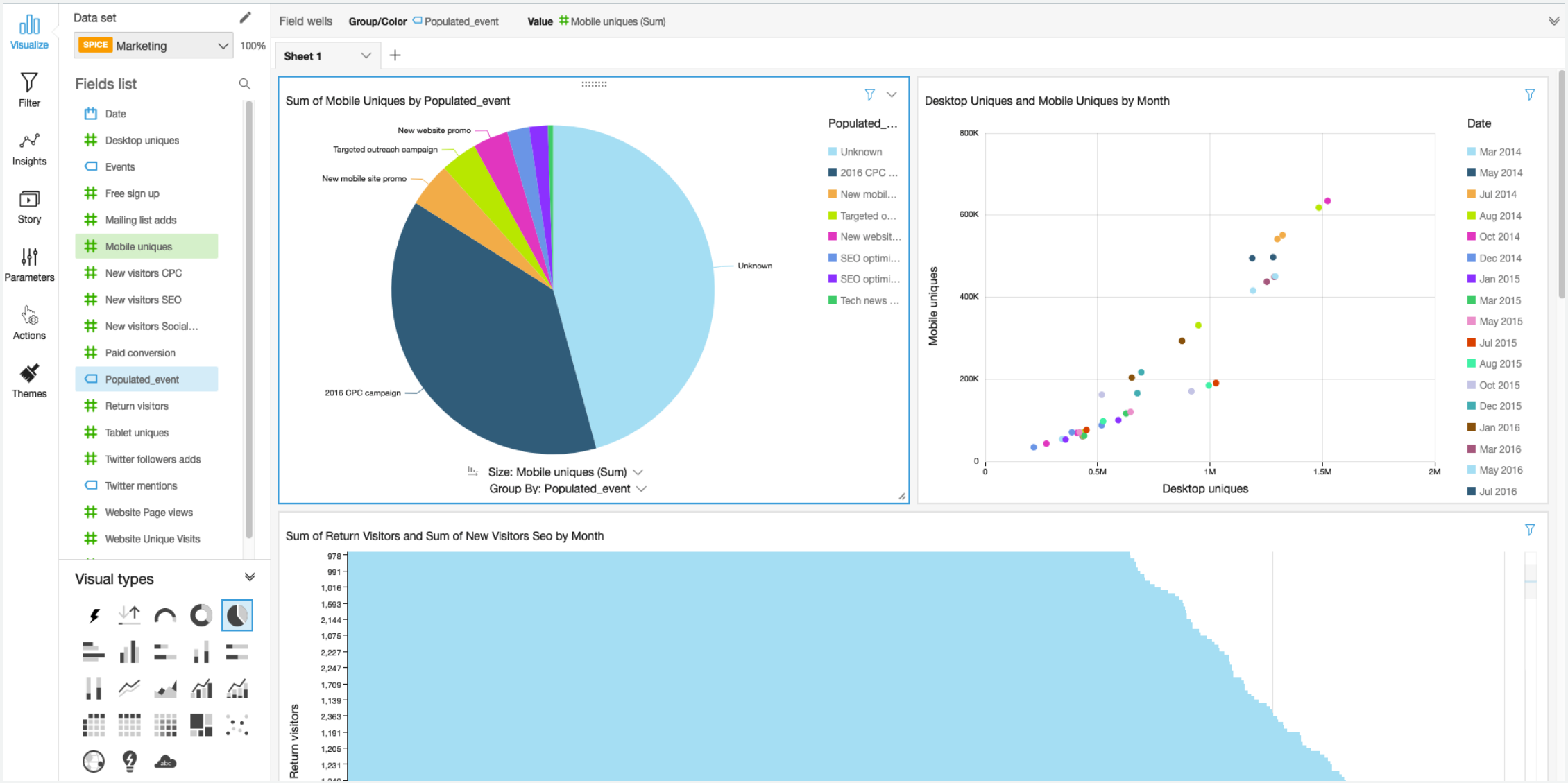

| | mlb_id | mlb_name | mlb_pos | mlb_team | mlb_team_long | bats | throws | birth_year | bp_id | bref_id | bref_name | cbs_id | cbs_ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | +5.0656000000000000e+05 | Alexi Amarista | 3B | SD | San Diego Padres | L | R | 1989 | +5.5889000000000000e+04 | amarial01 | Alexi Amarista | 1735053 | Alexi A |
| 2 | +4.5821000000000000e+05 | Alexi Casilla | 2B | TB | Tampa Bay Rays | S | R | 1984 | +4.5799000000000000e+04 | casilal01 | Alexi Casilla | 1103724 | Alexi C |
| 3 | +4.6839600000000000e+05 | Alexi Ogando | P | ATL | Atlanta Braves | R | R | 1983 | +4.9910000000000000e+04 | ogandal01 | Alexi Ogando | 1174266 | Alexi O |
| 4 | +4.6968600000000000e+05 | Alfredo Aceves | P | NYY | New York Yankees | R | R | 1982 | +4.6928000000000000e+04 | aceveal01 | Alfredo Aceves | 1638980 | Alfredo |
| 5 | +4.5162800000000000e+05 | Alfredo Figaro | P | TEX | Texas Rangers | R | R | 1984 | +5.2453000000000000e+04 | figaral01 | Alfredo Figaro | 1654334 | Alfredo |
| 6 | +5.5405400000000000e+05 | Alfredo Gonzalez | C | CWS | Chicago White Sox | R | R | 1992 | +6.9201000000000000e+04 | | | 2210083 | Alfredo |
| 7 | +5.0124500000000000e+05 | Alfredo Marte | LF | BAL | Baltimore Orioles | R | R | 1989 | +5.1755000000000000e+04 | marteal01 | Alfredo Marte | 1956711 | Alfredo |
| 8 | +4.3058000000000000e+05 | Alfredo Simon | P | CIN | Cincinnati Reds | R | R | 1981 | +4.5581000000000000e+04 | simonal01 | Alfredo Simon | 448968 | Alfredo |
| 9 | +4.5537800000000000e+05 | Ali Solis | C | LAD | Los Angeles Dodgers | R | R | 1987 | +5.2362000000000000e+04 | solisal01 | Ali Solis | 1946524 | Ali Solis |
| 10 | +4.8885200000000000e+05 | Allan Dykstra | 1B | TB | Tampa Bay Rays | L | R | 1987 | +5.7813000000000000e+04 | dykstal01 | Allan Dykstra | 1960245 | Allan D |
| 11 | +5.0180000000000000e+05 | Allen Craig | 1B | BOS | Boston Red Sox | R | R | 1984 | +5.1043000000000000e+04 | craigal01 | Allen Craig | 1661498 | Allen C |
| 12 | +5.4390300000000000e+05 | Allen Webster | P | PIT | Pittsburgh Pirates | R | R | 1990 | +5.8787000000000000e+04 | webstal01 | Allen Webster | 1799467 | Allen W |
| 13 | +6.0723700000000000e+05 | Amir Garrett | P | CIN | Cincinnati Reds | R | L | 1992 | +7.0946000000000000e+04 | | | 2053474 | Amir Ga |
| 14 | +4.4484300000000000e+05 | Andre Ethier | RF | LAD | Los Angeles Dodgers | L | L | 1982 | +4.5962000000000000e+04 | ethiean01 | Andre Ethier | 490390 | Andre E |
| 15 | +5.1658900000000000e+05 | Andre Rienzo | P | MIA | Miami Marlins | R | R | 1988 | +5.6638000000000000e+04 | rienzan01 | Andre Rienzo | 2027388 | Andre F |
| 16 | +5.9274300000000000e+05 | Andrelton Simmons | SS | LAA | Los Angeles Angels | R | R | 1989 | +6.7119000000000000e+04 | simmoan01 | Andrelton Simmons | 1918584 | Andrelt |
| 17 | +4.3321700000000000e+05 | Andres Blanco | 2B | PHI | Philadelphia Phillies | S | R | 1984 | +3.2835000000000000e+04 | blancan01 | Andres Blanco | 392116 | Andres |

# Analyze with Hadoop on Amazon EMR

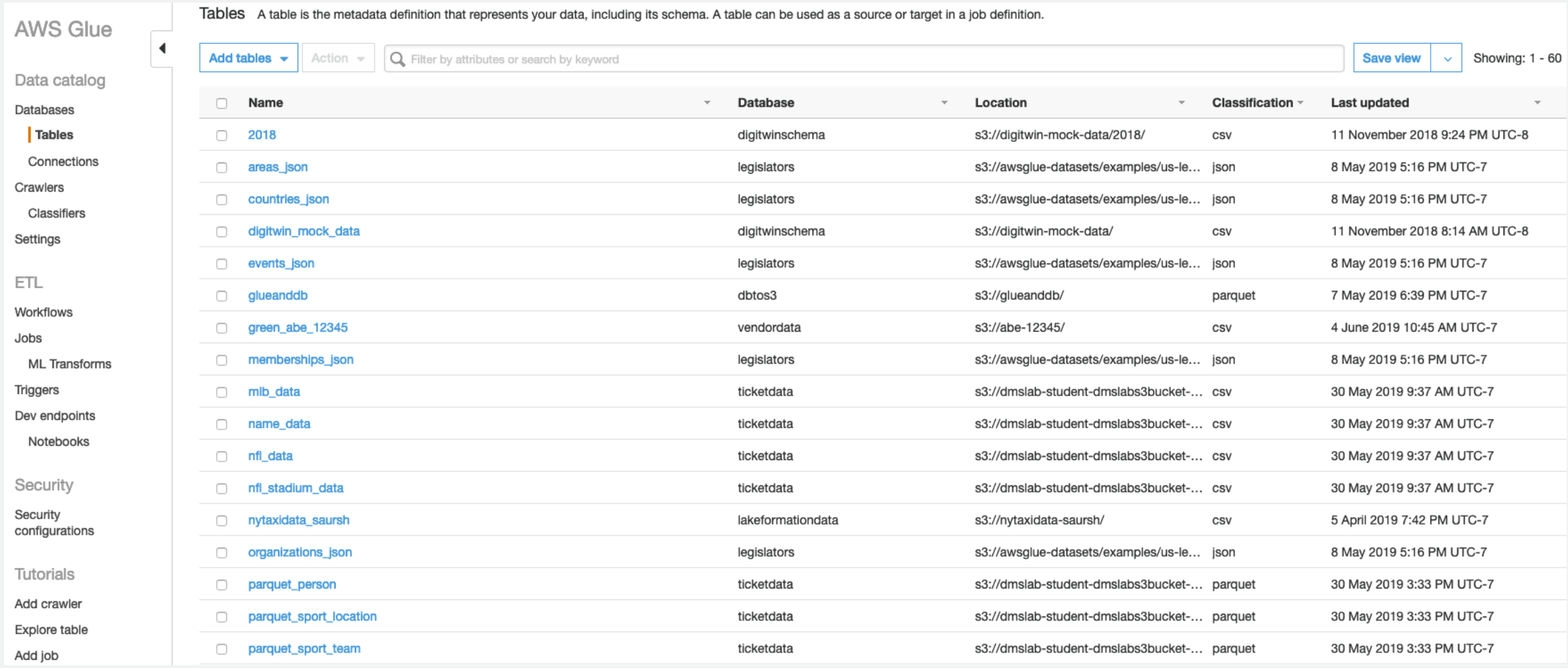



```scala
val movieLensHomeDir = "s3://emr.examples/movieLens/"

val movies = sc.textFile(movieLensHomeDir + "movies.dat").map { line =>
  val fields = line.split("::")
  // format: (movieId, movieName)
  (fields(0).toInt, fields(1))
}.collect.toMap

val ratings = sc.textFile(movieLensHomeDir + "ratings.dat").map { line =>
  val fields = line.split("::")
  // format: (timestamp % 10, Rating(userId, movieId, rating))
  (fields(3).toLong % 10, Rating(fields(0).toInt, fields(1).toInt, fields(2).toDouble))
}
```



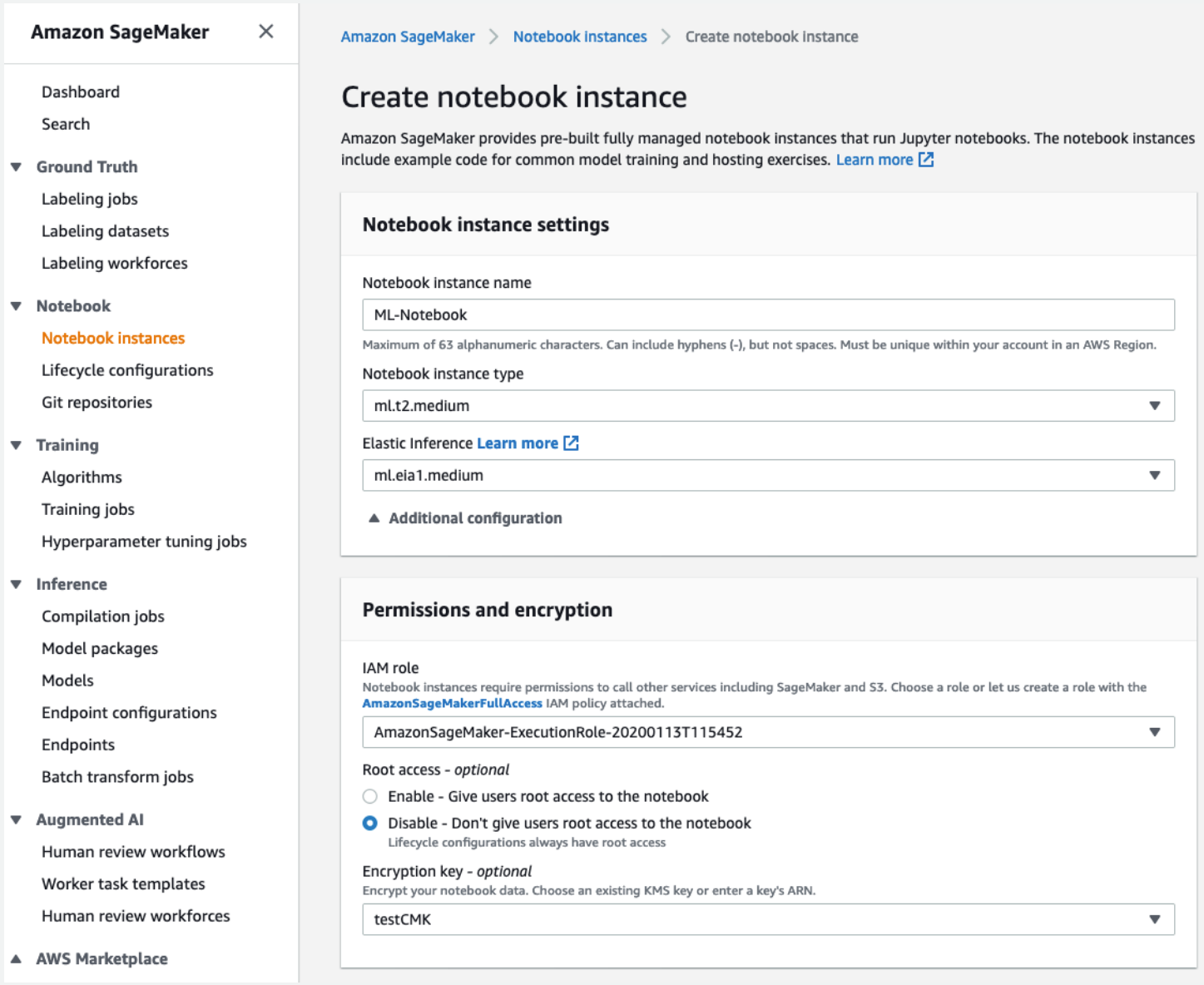


# Create a Central Data Catalog with AWS Glue




| Name | Database | Location | Classification | Last updated |
|---|---|---|---|---|
| 2018 | digitwinschema | s3://digitwin-mock-data/2018/ | csv | 11 November 2018 9:24 PM UTC-8 |
| areas_json | legislators | s3://awsglue-datasets/examples/us-le… | json | 8 May 2019 5:16 PM UTC-7 |
| countries_json | legislators | s3://awsglue-datasets/examples/us-le… | json | 8 May 2019 5:16 PM UTC-7 |
| digitwin_mock_data | digitwinschema | s3://digitwin-mock-data/ | csv | 11 November 2018 8:14 AM UTC-8 |
| events_json | legislators | s3://awsglue-datasets/examples/us-le… | json | 8 May 2019 5:16 PM UTC-7 |
| glueanddb | dbtos3 | s3://glueanddb/ | parquet | 7 May 2019 6:39 PM UTC-7 |
| green_abe_12345 | vendordata | s3://abe-12345/ | csv | 4 June 2019 10:45 AM UTC-7 |
| memberships_json | legislators | s3://awsglue-datasets/examples/us-le… | json | 8 May 2019 5:16 PM UTC-7 |
| mlb_data | ticketdata | s3://dmslab-student-dmslabs3bucket-… | csv | 30 May 2019 9:37 AM UTC-7 |
| name_data | ticketdata | s3://dmslab-student-dmslabs3bucket-… | csv | 30 May 2019 9:37 AM UTC-7 |
| nfl_data | ticketdata | s3://dmslab-student-dmslabs3bucket-… | csv | 30 May 2019 9:37 AM UTC-7 |
| nfl_stadium_data | ticketdata | s3://dmslab-student-dmslabs3bucket-… | csv | 30 May 2019 9:37 AM UTC-7 |
| nytaxidata_saursh | lakeformationdata | s3://nytaxidata-saursh/ | csv | 5 April 2019 7:42 PM UTC-7 |
| organizations_json | legislators | s3://awsglue-datasets/examples/us-le… | json | 8 May 2019 5:16 PM UTC-7 |
| parquet_person | ticketdata | s3://dmslab-student-dmslabs3bucket-… | parquet | 30 May 2019 3:33 PM UTC-7 |
| parquet_sport_location | ticketdata | s3://dmslab-student-dmslabs3bucket-… | parquet | 30 May 2019 3:33 PM UTC-7 |
| parquet_sport_team | ticketdata | s3://dmslab-student-dmslabs3bucket-… | parquet | 30 May 2019 3:33 PM UTC-7 |

# Train ML Models with Amazon SageMaker



# Load into Downstream Services

## Amazon Redshift

Run complex analytic queries against petabytes of structured data

## Amazon Aurora

A MySQL and PostgreSQL compatible relational database built for the cloud

## Amazon DynamoDB

A NoSQL database service that delivers consistent, single-digit millisecond latency at any scale.

## Amazon OpenSearch

Delivers real-time analytics capabilities alongside the availability, scalability, and security that production workloads require.

## Amazon SageMaker

fully managed service that provides every developer and data scientist with the ability to build, train, and deploy machine learning (ML) models quickly

# Thank you!

Paige Broderick
jbropaig@amazon.com