High Performance and Distributed Computing for Big Data

Unit 3: Big Data - Data Bases and Data Services

Jordi Mateo Fornés jordi.mateo@udl.cat

Universitat Rovira i Virgili and Universitat de Lleida

Today's lecture

1. **Big Data**: Data Bases and Data Services.
2. **HandsOn**: Exploring Dynamo DB and API Gateway
3. **HandsOn**: ETLs with AWS.
4. **HandsOn**: Playing with MongoDB.

Databases (SQL and NoSQL)

What is a SQL database?

In traditional **SQL databases**, data is stored in tables. *Each table has a schema that defines the columns and the data types.* Different tables can be related to each other using foreign keys.

```sql
CREATE TABLE patients (
    patient_id INT PRIMARY KEY,
    name VARCHAR(100),age INT);
```

```sql
CREATE TABLE medical_records (
    record_id INT PRIMARY KEY,
    patient_id INT,
    date DATE,
    diagnosis VARCHAR(100),
    FOREIGN KEY (patient_id)
    REFERENCES patients(patient_id)
);
```

While traditional SQL databases are robust, they have limitations. They are not optimized for handling large volumes of data, and scaling horizontally (across multiple servers) can be challenging.

Limitations

- Data Growth.
- Data Changes.
- Scalability.

What is a NoSQL database?

NoSQL databases are designed to *handle large volumes of data and to scale horizontally*. They are not based on the relational model. They are designed to be distributed and to be fault-tolerant.

Key Features

- **Flexible Data Storage**: Unlike traditional relational databases, NoSQL databases do not rely on a fixed schema. Instead, they allow data to be stored in various formats, making them suitable for structured, semi-structured, and unstructured data.
- **Distributed and Fault-Tolerant**: NoSQL databases are designed to be distributed across multiple servers. This ensures availability and reliability even if some servers go offline. *They can handle large volumes of data efficiently*.
- **Scalability**: You can add more servers to the cluster to handle increased workload. This approach is essential for modern web applications and big data scenarios.

Limitations

- **Lack of Structure**: Data can become untrustworthy and challenging to organize due to the absence of schemas.
- **Immaturity**: NoSQL DBs are relatively new, and their communities are smaller compared to SQL DBs.
- **Less Analytics Tools**: SQL DBs have a wide range of tools for analytics and reporting. NoSQL DBs have fewer options.

What are the types of NoSQL databases?

There are four main types of NoSQL databases:

Key-Value Stores

- Data is stored as key-value pairs.
- Examples: Amazon DynamoDB, Redis.

Document Stores

- Data is stored as documents (e.g., JSON objects).
- Examples: MongoDB, Couchbase.

Column-Family Stores

- Data is stored in columns rather than rows.
- Examples: Apache Cassandra, HBase.

Graph Databases

- Data is stored as nodes and edges.
- Examples: Neo4j, Amazon Neptune.

What is a Key-Value Store?

A **key-value store** is a NoSQL database that stores data as *key-value pairs*. Each key is unique and maps to a value. The value can be a string, number, boolean, array, or another object.

```
[{
    "key": "patient_id",
    "value": "12345"
  },
  {
    "key": "name",
    "value": "John Doe"
  },
  {
    "key": "age",
    "value": 45
  }]
```

Advantages

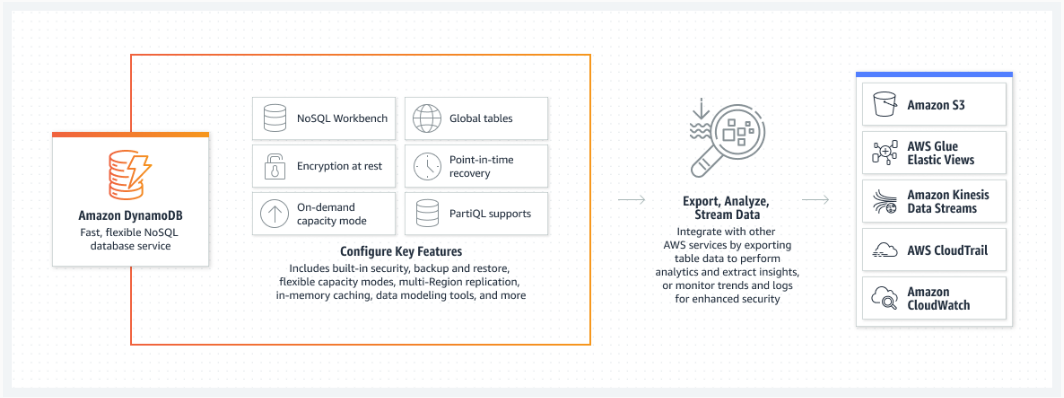
- **Simplicity**: Easy to use and understand.
- **Read/Write Performance**: Fast read and write operations based on the key.

Limitations

- **Query Capabilities**: Limited to querying by key.
- **Complex Relationships**: Difficult to model complex relationships between data (*joins* or *aggregations*).

What is DynamoDB?

Amazon DynamoDB is a fully managed proprietary NoSQL database service that supports key-value and document data structures. It is highly available, eventual consistent, and scalable.


Amazon DynamoDB
Fast, flexible NoSQL database service
NoSQL Workbench
Global tables
Encryption at rest
Point-in-time recovery
On-demand capacity mode
PartiQL supports
Configure Key Features
Includes built-in security, backup and restore, flexible capacity modes, multi-Region replication, in-memory caching, data modeling tools, and more
Export, Analyze, Stream Data
Integrate with other AWS services by exporting table data to perform analytics and extract insights, or monitor trends and logs for enhanced security
Amazon S3
AWS Glue Elastic Views
Amazon Kinesis Data Streams
AWS CloudTrail
Amazon CloudWatch

What is a document store?

A document store is a **JSON object** that contains *key-value pairs*. The document is not opaque. It is self-describing, meaning that the data and the schema are stored together.

```
{
  "patient": {
    "id": "12345",
    "name": "John Doe",
    "diagnosis": "Hypertension"
  }, "lab_results": [
    {
      "test_name": "Blood Pressure",
      "value": "140/90 mmHg",
      "date": "2024-03-28"
    }, # more results...
  ], # more patients...}
```

Advantages

- **Flexibility**: Store complex data structures in a single document.
- **Query Capabilities**: Query the data based on the values of the fields.

Limitations

- Complex Queries
- Tuning
- Lack of transactions
- Data duplication

What is MongoDB?

MongoDB is a popular open-source document store that stores data as JSON-like documents. It is designed for flexibility and scalability.

```
db.createCollection("patients")
db.patients.insertOne({
  "patient_id": 12345,
  "name": "John Doe",
  "diagnosis": "Hypertension"
})
db.createCollection("lab_results")
db.lab_results.insertOne({
  "patient_id": 12345,
  "test_name": "Blood Pressure",
  "value": "140/90 mmHg",
  "date": "2024-03-28"
})
```

```
db.patients.find("name": "John Doe")
db.lab_results.aggregate([
  { $lookup: {
      from: "patients",
      localField: "patient_id",
      foreignField: "patient_id",
      as: "patient"
    }
  },{
    $match: {
      "patient.name": "John Doe"
    }
  }
])
```

What is a column-family store?

A **column-family** store is a NoSQL database that stores data in columns rather than rows. Each row is a key-value pair, where the key is the row key and the value is a set of columns. Each column has a name and a value.

```
{
  "patient_id": {
    "name": {"value": "John Doe",
    "timestamp": 1630454873},
    "age": {"value": 45,
    "timestamp": 1630454873},
    "diagnosis": {"value": "Hypertension",
    "timestamp": 1630454873}
  }
}
```

Advantages

- The column-family store is optimized for read and write operations on a large scale.
- Restricting and application to a only certain column families (access control).
- Store local groups of data together.

Limitations

- **Complexity**.
- **Data Duplication**.
- **Query Capabilities**.

What is a graph database?

A **graph database** is a NoSQL database that stores data as nodes and edges. *Nodes represent entities*, and *edges represent relationships between entities*. Each node has properties, and each edge has a type and direction.

```
{
  "nodes": [
    {"id": "1", "label": "Patient", "properties": {"name": "John Doe", "age": 45}},
    {"id": "2", "label": "Diagnosis", "properties": {"name": "Hypertension"}}
  ],
  "edges": [
    {"source": "1", "target": "2", "type": "HAS_DIAGNOSIS",
    "properties": {"date": "2024-03-28"}}
  ]
}
```

HandsOn: Exploring Dynamo DB and API Gateway

Genomic Data

The provided data represents genomic variants with the following fields: *ID, Chromosome, Position, Reference Allele, Alternate Allele, Genotype, Quality, and Depth*. Each row represents a variant.

```
ID,Chromosome,Position,Reference_Allele,Alternate_Allele,Genotype,Quality,Depth
1,1,100001,A,T,AA,30,50
2,1,150002,C,G,CC,28,45
3,2,500003,G,A,GG,35,60
4,2,750004,T,C,TT,32,55
5,3,300005,A,G,AG,25,40
6,3,450006,C,T,CC,28,48
7,4,900007,G,T,GT,31,52
8,4,1200008,A,C,AC,27,44
9,5,600009,T,G,TG,33,58
10,5,900010,C,A,CA,29,46
```

Data Ingestion

1. Go to the S3 console using the AWS Management Console.
2. Click on “Create bucket”.

- Bucket name: hdbc-yourname-genomics
- Region: US East (N. Virginia)

3. Click on “Create bucket”.
4. Click on the bucket name.
5. Click on “Upload”.
6. Select the genomic data file and click on “Upload”.

Data Storage with DynamoDB

1. Go to the DynamoDB console using the AWS Management Console.
2. Click on Imports from S3.
    - Import options:
        - Source S3 bucket: hdbc-yourname-genomics
        - Input file format: CSV
        - CSV header: Use first row for column names
        - CSV delimiter: Comma
    - Table details:
        - Table name: genomics
        - Primary key: ID
    - Table settings: Keep the default settings
3. Click on Import.
4. Wait for the import to complete.

Viewing the Data in DynamoDB

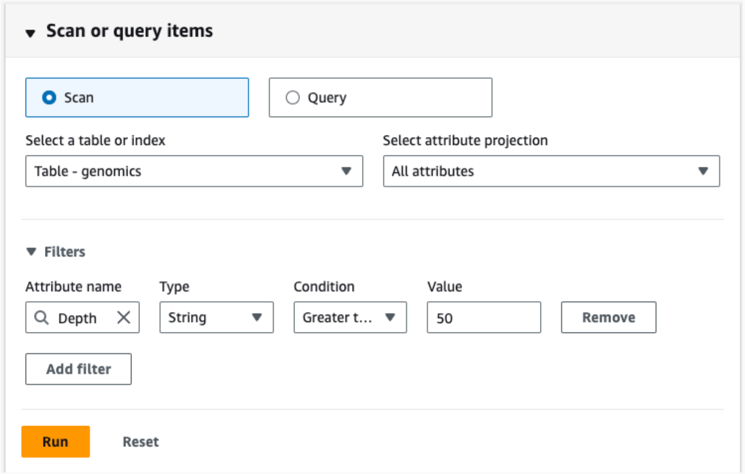
Steps

1. Go to the DynamoDB console using the AWS Management Console.
2. Click on the “Tables” tab.
3. Click on the “genomics” table.
4. Click on the “Items” tab to view the data.

Example Query

Retrieve all items with a depth greater than 50.

How to Query



Scan or query items

[✓ Scan] Query

Select a table or index: Table - genomics

Select attribute projection: All attributes

Filters

Attribute name	Type	Condition	Value
Depth	String	Greater t...	50

Data Retrieval with AWS Lambda and Amazon API Gateway

Storing the data in DynamoDB is the first step. But we also want to access it and disseminate it. There is a lot of options to do this. *We can create a EC2 instance with jupyter notebook and access the data from there.* We can create a web application using Flask and access the data from there. **Or we can create a RESTful API using AWS Lambda and Amazon API Gateway to retrieve the data from the DynamoDB table**.

Goal

We are going to configure a RESTful API that retrieves all variants from the DynamoDB table and returns them as JSON. Besides, we are going to improve the API to accept **query parameters** to **filter** the data based on the **chromosome**.

- **GET** */genomics*: Retrieve all variants.
- **GET** */genomics?chromosome=1*: Retrieve all variants on chromosome 1.

AWS Lambda Function (Retrieve All Variants)

Steps

1. Go to the Lambda console using the AWS Management Console.
2. Click on “Create function”.

- Function name: genomics-retrieval
- Runtime: Python 3.12
- Architecture: x86_64
- Change default execution role: LabRole
- Click on “Create function”.

```python
import json
import boto3
from decimal import Decimal

def lambda_handler(event, context):
    try:
        dynamodb = boto3.resource('dynamodb')
        table = dynamodb.Table('genomics')
        response = table.scan()
        items = response['Items']
        for item in items:
            for key, value in item.items():
                if isinstance(value, Decimal):
                    item[key] = float(value)
        return {
            'statusCode': 200,
            'body': json.dumps(items)
        }
    except Exception as e:
        return {
            'statusCode': 500,
            'body': f'Error retrieving data: {str(e)}'
        }
```

AWS API Gateway

AWS API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. API Gateway acts as a front door for applications to access data, business logic, or functionality from your back-end services (*CRUD operations, Lambda functions, etc.*).

1. Go to the API Gateway console using the AWS Management Console.
2. Create a REST API.
   - **API name**: genomics-api
   - **Endpoint type**: Regional
   - Click on “Create API”.
3. Create a resource.
   - **Resource name**: genomics
   - Click on “Create resource”.
4. Create a method.
   - **Method**: GET
   - **Integration type**: Lambda function
   - **Lambda function**: genomics-retrieval
   - Click on “Save”.
5. Deploy the API.
   - Deployment stage: [New stage]
   - Stage name: prod
   - Click on “Deploy”.
6. Click on the “Invoke URL” to test the API.

Making the response more readable

If you test the API now, you will see that the response is not very readable. We can improve it by adding a mapping template to the Integration Response. This template will map the response from the Lambda function to a more readable JSON format.

1. Go to Integration Response.
2. Click on “Mapping templates”.
3. Add a new mapping template.
   - **Content-Type**: application/json
   - **Generate template**: Method response passthrough

```
{
  "statusCode": $input.json('$.statusCode'),
  "headers": {
    "Content-Type": "application/json"
  },
  "body": $input.path('$.body')
}
```

Improving the Lambda Function to Retrieve Data

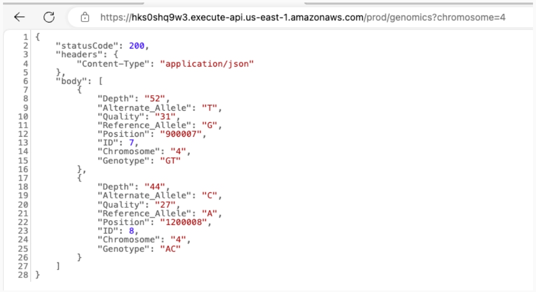
```python
import json
import boto3
from boto3.dynamodb.conditions import Key
from decimal import Decimal

def lambda_handler(event, context):
    try:
        dynamodb = boto3.resource('dynamodb')
        table = dynamodb.Table('genomics')
        response = table.scan()
        if  'chromosome' in event:
            chromosome = event['chromosome']
            if not chromosome == '':
                response = table.scan(
                    FilterExpression=
                    Key('Chromosome').eq(chromosome)
                 )
        items = response['Items']
        for item in items:
            for key, value in item.items():
                if isinstance(value, Decimal):
                    item[key] = float(value)
        return {
            'statusCode': 200, 'body': json.dumps(items)
        }
    except Exception as e:
        return {
            'statusCode': 500,
            'body': f'Error retrieving data: {str(e)}'
        }
```

Steps

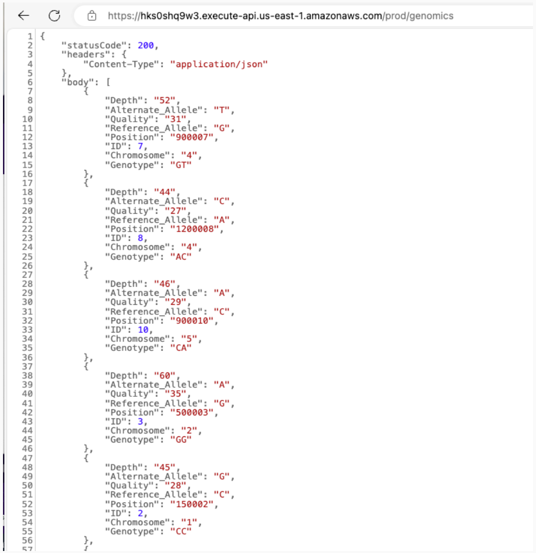
1. Go to the API Gateway console using the AWS Management Console.
2. Click on the *genomics* resource and **GET** method.
3. Click on *Integration Request* and *Mapping templates*.
4. Add a new mapping template.
    - **Content-Type**: application/json
    - **Generate template**: Method request passthrough

```
{
  "chromosome":
    "$input.params('chromosome')"
}
```

Testing the API


https://hks0shq9w3.execute-api.us-east-1.amazonaws.com/prod/genomics?chromosome=4

```
{
    "statusCode": 200,
    "headers": {
        "Content-Type": "application/json"
    },
    "body": [
        {
            "Depth": "52",
            "Alternate_Allele": "T",
            "Quality": "31",
            "Reference_Allele": "G",
            "Position": "900007",
            "ID": 7,
            "Chromosome": "4",
            "Genotype": "GT"
        },
        {
            "Depth": "44",
            "Alternate_Allele": "C",
            "Quality": "27",
            "Reference_Allele": "A",
            "Position": "1200008",
            "ID": 8,
            "Chromosome": "4",
            "Genotype": "AC"
        }
    ]
}
```


https://hks0shq9w3.execute-api.us-east-1.amazonaws.com/prod/genomics

```
{
    "statusCode": 200,
    "headers": {
        "Content-Type": "application/json"
    },
    "body": [
        {
            "Depth": "52",
            "Alternate_Allele": "T",
            "Quality": "31",
            "Reference_Allele": "G",
            "Position": "900007",
            "ID": 7,
            "Chromosome": "4",
            "Genotype": "GT"
        },
        {
            "Depth": "44",
            "Alternate_Allele": "C",
            "Quality": "27",
            "Reference_Allele": "A",
            "Position": "1200008",
            "ID": 8,
            "Chromosome": "4",
            "Genotype": "AC"
        },
        {
            "Depth": "46",
            "Alternate_Allele": "A",
            "Quality": "29",
            "Reference_Allele": "C",
            "Position": "900010",
            "ID": 10,
            "Chromosome": "5",
            "Genotype": "CA"
        },
        {
            "Depth": "60",
            "Alternate_Allele": "A",
            "Quality": "35",
            "Reference_Allele": "G",
            "Position": "500003",
            "ID": 3,
            "Chromosome": "2",
            "Genotype": "GG"
        },
        {
            "Depth": "45",
            "Alternate_Allele": "G",
            "Quality": "28",
            "Reference_Allele": "C",
            "Position": "150002",
            "ID": 2,
            "Chromosome": "1",
            "Genotype": "CC"
        },
        {
```

HandsOn: ETLs with AWS

What is an ETL?

An **ETL pipeline** is a set of processes that extract data from a source, transform it, and load it into a destination. ETL stands for Extract, Transform, Load.

Data Ingestion

- E: Get data from one or more sources.
- T: Clean, filter, and transform the data.
- L: Load the transformed data into a destination.

Data Transformation

- E: Retrieve raw data from the data lake.
- T: Apply business rules, aggregate, join, or filter the data.
- L: Store data back into the data lake.

Data Quality and Governance

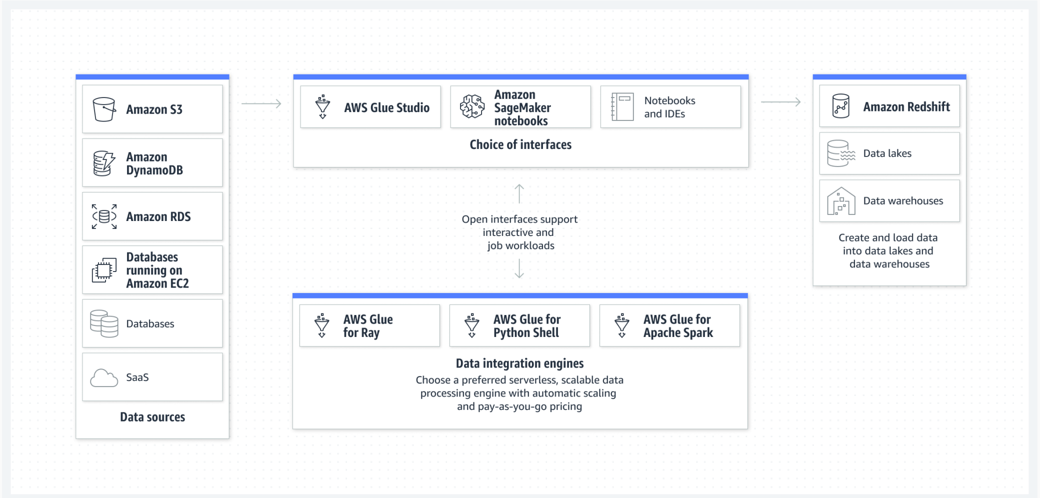
- E: Fetch from different sources.
- T: Validate, standardize, and cleanse the data.
- L: Ensure high-quality data for downstream processes.

Incremental Updates

- E: Identify new or changed data.
- T: Apply updates incrementally.
- L: Append or update existing data in the data lake.

AWS Glue

AWS Glue is a fully managed extract, transform, and load (ETL) service that makes it easy to prepare and load data for analytics. You can create and run ETL jobs with a few clicks in the AWS Management Console.


Amazon S3
Amazon DynamoDB
Amazon RDS
Databases running on Amazon EC2
Databases
SaaS
Data sources
AWS Glue Studio
Amazon SageMaker notebooks
Notebooks and IDEs
Choice of interfaces
Open interfaces support interactive and job workloads
AWS Glue for Ray
AWS Glue for Python Shell
AWS Glue for Apache Spark
Data integration engines
Choose a preferred serverless, scalable data processing engine with automatic scaling and pay-as-you-go pricing
Amazon Redshift
Data lakes
Data warehouses
Create and load data into data lakes and data warehouses

Components of AWS Glue

1. **Crawlers**: Automatically discover the schema of your data and create metadata tables in the AWS Glue Data Catalog.
2. **Jobs**: Run ETL jobs to extract, transform, and load data from a source to a destination.
3. **Triggers**: Schedule jobs to run at specific times or in response to events.
4. **Workflows**: Create complex ETL workflows with dependencies between jobs.
5. **Data Catalog**: Store metadata about your data, such as tables, databases, and schemas.

AWS Glue: Creating a Crawler (I)

To create a crawler in AWS Glue, you need to specify the data source, the database, and the classifier. The crawler will automatically discover the schema of your data and create metadata tables in the AWS Glue Data Catalog.

- Go to the AWS Glue console using the AWS Management Console.
- Click on **Crawlers** and then **Add crawler**.
- *Crawler name*: **genomics-crawler**
- Next
- *Is your data already mapped to Glue tables?* **No**
- Add a data store.
- Add a new classifier.
- *IAM Role*: **LabRole**
- Add database.
- Select the database and click on “Next”.
- Create crawler.
- Run crawler.

AWS Glue: Creating a Crawler (II)

Datasource Configuration

- Data source: S3
- S3 path: s3://hdbc–genomics
- Subsequent crawls: Crawl all sub-folders

Database Configuration

- Database name: genomics
- Table prefix: hdbc-<yourname}-genomics

Classifier Configuration

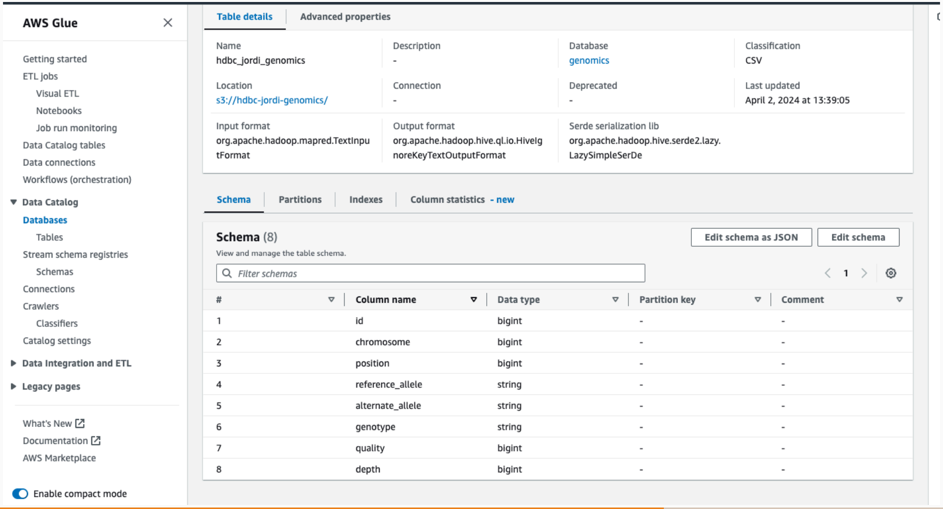
- Classifier name: genomics-classifier
    - Classifier type: CSV
    - Keep the rest as default

Classifier schema

```
{
"Type": "CSV",
"Name": "genomics-classifier",
"Delimiter": ",",
"Header": "Use the first
    row as column names",
"QuoteSymbol": "\"",
"ContainsHeader": "Present",
"DisableValueTrimming": false,
"AllowSingleColumn": false
}
```

Running the Crawler

Once the crawler has finished running, you can view the table in the Glue Data Catalog. The table should have the following schema:



Name hdbc_jordi_genomics	Description -	Database genomics	Classification CSV
Location s3://hdbc-jordi-genomics/	Connection -	Deprecated -	Last updated April 2, 2024 at 13:39:05
Input format org.apache.hadoop.mapred.TextInputFormat	Output format org.apache.hadoop.hive.ql.io.HiveIgnoreKeyTextOutputFormat	Serde serialization lib org.apache.hadoop.hive.serde2.lazy.LazySimpleSerDe	

Schema (8)

View and manage the table schema.

#	Column name	Data type	Partition key	Comment
1	id	bigint	-	-
2	chromosome	bigint	-	-
3	position	bigint	-	-
4	reference_allele	string	-	-
5	alternate_allele	string	-	-
6	genotype	string	-	-
7	quality	bigint	-	-
8	depth	bigint	-	-

AWS Glue: Creating a Job (I)

To create a job in AWS Glue, you need to specify the data source, the data target, and the transformation script.

Steps

- Go to the AWS Glue console using the AWS Management Console.
- Click on **Creatre job** from visual ETL.
  - **Job name**: **genomics-etl**
  - Add nodes.
  - **IAM role**: LabRole
- Click on **Save** and then **Run job**.

Nodes

1. **Data source**: DynamoDB
   - Choose from the AWS Glue Data Catalog
   - **Database**: genomics
   - **Table**: hdbc–genomics
2. **Data transformation**: filter
   - **Name**: quality-filter
   - **Filter**: Global AND
   - **Condition**: Quality > 30
3. **Data target**: S3
   - **Name**: genomics-csv-filtered
   - **Format**: CSV
   - **Compression**: None
   - **Target path**: s3://hdbc–genomics-filtered/
   - **Data Catalog**: Do not update the data catalog

AWS Glue: Creating a Job (II)

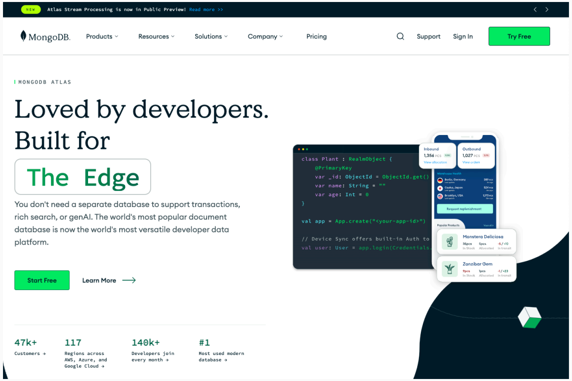
Once the job has finished running, you can view the filtered data in the S3 bucket. The data should be filtered based on the quality field, and the rows under a certain threshold should be discarded.

Take a look at the data in the S3 bucket to see the results of the transformation.

Hands On: Playing with MongoDB

Mongo Atlas

Mongo Atlas is a cloud-based MongoDB service. It offers a free tier M0 cluster which is a great way to get started with MongoDB. In this hands-on, we will create a free tier M0 cluster and connect to it using MongoDB Compass. MongoDB Atlas: https://www.mongodb.com/cloud/atlas



MONGODB ATLAS

Loved by developers. Built for The Edge

You don't need a separate database to support transactions, rich search, or genAI. The world's most popular document database is now the world's most versatile developer data platform.

47k+ Customers →

117 Regions across AWS, Azure, and Google Cloud →

140k+ Developers join every month →

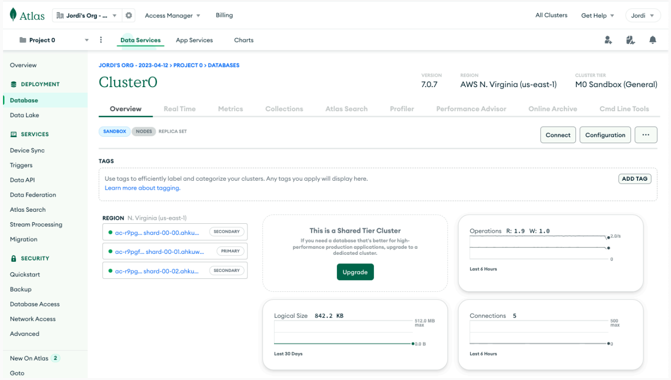
#1 Most used modern database →

```
class Plant : RealmObject {
    @PrimaryKey
    var _id: ObjectId = ObjectId.get()
    var name: String = ""
    var age: Int = 0
}

val app = App.create("<your-app-id>")

// Device Sync offers built-in Auth to
val user: User = app.login(Credentials.
```

Click on **Try Free** to create a free account. *You can sign up using your Google account.*

Free Tier M0 Cluster

- The M0 free tier provides a **sandbox environment**. With limited resources (*512 MB storage, shared RAM*). This is great for **learning** and *small projects*.



JORDI'S ORG - 2023-04-12 > PROJECT 0 > DATABASES

Cluster0

VERSION 7.0.7 REGION AWS N. Virginia (us-east-1) CLUSTER TIER M0 Sandbox (General)

TAGS

Use tags to efficiently label and categorize your clusters. Any tags you apply will display here.
Learn more about tagging.

REGION N. Virginia (us-east-1)

- ac-r9pg... shard-00-00.ahku... SECONDARY
- ac-r9pgf... shard-00-01.ahkuw... PRIMARY
- ac-r9pg... shard-00-02.ahku... SECONDARY

This is a Shared Tier Cluster

If you need a database that's better for high-performance production applications, upgrade to a dedicated cluster.

Operations R: 1.9 W: 1.0 — 2.0/s, 0 — Last 6 Hours

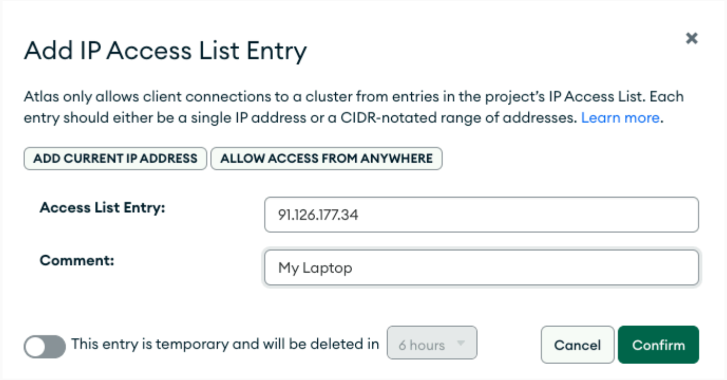
Logical Size 842.2 KB — 512.0 MB max, 0.0 B — Last 30 Days

Connections 5 — 500 max, 0 — Last 6 Hours

Figure 1: View info

Network Configuration

1. Click on **Network Access** on the left-hand side menu. And then click on **Add IP Address**.
2. Add your current IP address to the whitelist. This will allow you to connect to the cluster from your local machine.



Add IP Access List Entry

Atlas only allows client connections to a cluster from entries in the project's IP Access List. Each entry should either be a single IP address or a CIDR-notated range of addresses. Learn more.

Access List Entry: 91.126.177.34

Comment: My Laptop

This entry is temporary and will be deleted in 6 hours

Connect to the Cluster

1. Click on **Database** and then click on **Connect**.
2. Create a MongoDB user. This user will be used to connect to the cluster.

- Username: `your-username`
- Password: `your-password`

3. Click on **Choose a connection method**.

- Select Compass.
- *Follow to install the client on your machine.*

4. Click on **Connect**.

Connect to the Cluster using MongoDB Compass

- Open MongoDB Compass.
- Click on **New Connection**.
- Enter the connection string and click on **Connect**.
- Replace `<password>` with your password.

Operations with MongoDB

We are going to create a database called `hospital-data` and two collections: `patients` and `diagnoses`. We will import the data from the JSON files provided.

- Create a database called `hospital-data`.
- Create a collection called `patients`.
- Create a collection called `diagnoses`.
- Import the data from the JSON files provided.

Query the Database

Query 1: Find All Patients with Age Greater Than 50

```
{"age": {"$gt": 50 }}
```

Query 2: Find All Patients with Hipertention Diagnosis

```
{ diagnoses: {$elemMatch: {code: "I10"}}}
```

Aggregation 1: Count Patients with Each Diagnosis

```
[
  {
    $lookup: {
        from: "diagnoses",
        localField: "_id",
        foreignField: "patient_id",
        as: "results"
    }
  },
    {
        $unwind: "$results"
    },
    {
        $group: {
            _id: "$results.code",
            count: { $sum: 1 }
        }
    }
]
```

Aggregation 2: Count Patients with Each Diagnosis and Age Greater Than 50

```
[
  {
    $lookup: {
        from: "diagnoses",
        localField: "_id",
        foreignField: "patient_id",
        as: "results"
    }
  },
    {
        $unwind: "$results"
    },
    {
        $match: {
            age: { $gt: 50 }
        }
    },
    {
        $group: {
            _id: "$results.code",
            count: { $sum: 1 }
        }
    }
]
```

Conclusions

Conclusions

- **NoSQL databases** are designed to handle large volumes of data and to scale horizontally. They are optimized for read and write operations on a large scale.
- AWS provides a range of NoSQL databases, such as **DynamoDB**, **DocumentDB**, and **Neptune**, that are fully managed and scalable.
- AWS Glue is a fully managed extract, transform, and load (ETL) service that makes it easy to prepare and load data for analytics. You can create and run ETL jobs with a few clicks in the AWS Management Console.
- We create a MongoDB cluster using **MongoDB Atlas** and connect to it using **MongoDB Compass**. We create a database and collections and import data from JSON files. We perform queries and aggregations to analyze the data.

To Recap

- During these weeks, we have covered EC2, S3, Networking, Lambda, Elastic Map Reduce, SQL and NoSQL databases, and ETLs with AWS Glue.
- I hope you have learned a lot and enjoyed this journey through cloud and big data technologies.
- Remember that practice is key to mastering these technologies. Keep practicing and exploring new use cases to deepen your knowledge.

That’s all

Thanks for your attention!

Questions?