Angel Miguel & Sara Barsan

ARANGODB

Phase 1: Research & Documentation

The first phase of the project focuses on thoroughly researching ArangoDB, documenting its key features, characteristics, and architecture. This will give a clear understanding of how ArangoDB fits into the NoSQL database landscape and help in later practical implementation.

1. Characteristics of NoSQL Databases:

A nonSQL database is a type of non relational database, meaning it is able to store the same data as a RDBMS (Relational DataBase Management System) but it changes in the structure it uses.

It does not require a scheme offering faster scalability, normally using big chunks of non structured data.

It is also a distributed database granting avaliability and consistency of data, if some of the servers disconnect the database can still operate.

Because of these factors Cloud, Big Data and web / mobile apps are key markets in which noSQL databases are the popular choice.

---

1. Relational Database :
RDBMS stands for Relational Database Management Systems. It is most popular database. In it, data is store in the form of row that is in the form of tuple. It contain numbers of table and data can be easily accessed because data is store in the table. This Model was proposed by E.F. Codd.

2. NoSQL :
NoSQL Database stands for a non-SQL database. NoSQL database doesn't use table to store the data like relational database. It is used for storing and fetching the data in database and generally used to store the large amount of data. It supports query language and provides better performance.

Difference between Relational database and NoSQL :

Relational Database	**NoSQL**
It is used to handle data coming in low velocity.	It is used to handle data coming in high velocity.
It gives only read scalability.	It gives both read and write scalability.
It manages structured data.	It manages all type of data.

Data arrives from one or few locations.	Data arrives from many locations.
It supports complex transactions.	It supports simple transactions.
It has single point of failure.	No single point of failure.
It handles data in less volume.	It handles data in high volume.
Transactions written in one location.	Transactions written in many locations.
support ACID properties compliance	doesn’t support ACID properties
Its difficult to make changes in database once it is defined	Enables easy and frequent changes to database
schema is mandatory to store the data	schema design is not required
Deployed in vertical fashion.	Deployed in Horizontal fashion.

ADVANTAGES NOSQL

- Can support easy updates to schemas
- Can process large volumes of data at high speed
- Storage is scalable
- Can store structured, semi-structured, and structured

DISADVANTAGES NOSQL

Lack of SQL

Lack of Standarization

No ACID so data is less trustworthy

Lacks JOINs

2. Types of NoSQL Databases:

- **What type is ArangoDB?**
    - **ArangoDB is a Multi-model Database**, meaning it supports multiple types of data models within a single database engine. Unlike traditional NoSQL databases that specialize in one type (e.g., document stores like MongoDB or graph databases like Neo4j), ArangoDB integrates:

1. **Document Store** – Supports JSON-based documents, similar to MongoDB.
2. **Key-Value Store** – Allows simple key-value storage, comparable to Redis.
3. **Graph Database** – Provides native graph processing capabilities, akin to Neo4j.

This multi-model approach allows developers to use different data representations without needing multiple databases.

- Discuss why multi-model databases are beneficial for applications that require flexible data storage and complex relationships (e.g., social networks, recommendation engines).
    - **Document Store**: Uses collections to store JSON documents, enabling schema-free storage with indexing for fast queries.
    - **Key-Value Store**: Provides efficient key-value lookups where a document can be retrieved using a unique key.
    - **Graph Database**: Supports vertices (nodes) and edges (relationships) with powerful traversal and pattern-matching queries via the ArangoDB Query Language (AQL).

ArangoDB unifies these models with **a single query language (AQL)** and **one database engine**, simplifying data management and reducing operational overhead

Benefits of Multi-Model Databases

- **Flexibility in Data Modeling**: Supports different structures (documents, graphs, key-value pairs) within the same system.
- **Reduced Complexity**: No need for separate databases for different models, simplifying architecture and maintenance.

- **Optimized Performance**: Enables efficient data retrieval by leveraging the best model for a given use case.
- **Ideal for Complex Relationships**: Essential for applications with interconnected data, such as social networks or fraud detection.

Use Cases for ArangoDB

1. **Social Networks** – Managing user connections and interactions efficiently using a graph structure.
2. **Recommendation Engines** – Storing user preferences as key-value pairs while analyzing relationships through graph queries.
3. **Fraud Detection** – Identifying suspicious transactions by analyzing connections between users and financial activities.
4. **Knowledge Graphs** – Organizing large datasets with relationships to improve search and discovery.
5. **Real-Time Analytics** – Combining key-value lookups with document storage for fast analytics.

By integrating multiple data models, ArangoDB provides a **scalable and high-performance** solution for applications requiring flexible data storage and complex relationships.

3. Elements of NoSQL Databases:

ArangoDB Data Structure:

ArangoDB stores graphs and objects in JSON files and they organize in collections and databases

Each record is stored as a JSON object (document).

Each key value pair is called an attribute (also called field or property), comprised of value and name.

The hierarchy that data is organized in is **documents** (data records) in **collections**, and collections in **databases**.

ArangoDB Query language:

AQL (ArangoDB Query Language) is the language used to query documents, graphs and key-value stores

It is a Declarative Language with a focus in client Independence, which means that it doesn't matter what programming language you use the syntax will remain the same.

It is a DML (Data Manipulation Language) exclusively not a DDL (Data Definition Language). Databases in ArangoDB are typically created through administrative tools or REST API*, not through AQL queries.

Indexing Replication and Consistency Models:

· **Indexing** in ArangoDB helps speed up queries by allowing the database to quickly locate relevant data in a collection, there are various types of indexes like hash, full-text, and geo indexes.

Types of Indexes in ArangoDB:

- **Primary Index**: Automatically created on the _key field of every collection, ensuring uniqueness for each document in that collection.
- **Hash Index**: Used for fast lookups on fields. Suitable for equality checks.
- **Skiplist Index**: Useful for range queries (e.g., greater than, less than).
- **Fulltext Index**: Allows for efficient text search within a collection.
- **Geo Index**: Used for geospatial data to perform queries like distance-based searches.
- **Persistent Index**: Stores index data on disk and is used for larger collections.

· **Replication** ensures that data is duplicated across servers for high availability. ArangoDB supports master-slave replication and active-active replication, with options to pull and push data between servers.

· **Master-Slave Replication**: One server (master) handles all write operations, and the others (slaves) replicate the data from the master.

· **Active-Active Replication**: Multiple servers handle both reads and writes, and the data is synchronized between these servers.

· **Consistency Models** define how data consistency is handled in distributed setups. ArangoDB offers tunable consistency, allowing users to balance between strong consistency (ensuring the most recent data) and eventual consistency (prioritizing availability and speed). You can configure consistency settings in both replication and in individual queries.

· **Strong Consistency**: Ensures that when a write is confirmed, all subsequent reads will see the result of that write. It can affect performance due to synchronization across nodes.

· **Eventual Consistency**: Allows for a faster write operation and eventual synchronization of data across nodes. This model is typically used in distributed systems where availability is prioritized over immediate consistency.

· **Tunably Consistent**: ArangoDB provides a tunable consistency model where you can choose the level of consistency you want for reads and writes. You can configure it to be strongly consistent or eventually consistent.

4. Database Management Systems (DBMS) for ArangoDB:

- **ArangoDB Management System:**

1. Transaction Handling in Multi-Model Architecture

ArangoDB is a **multi-model NoSQL database** supporting **document, key-value, and graph data models**. Transactions in ArangoDB work across these models seamlessly:

- **Single-Document Transactions**: Atomic updates to individual documents using MVCC (Multi-Version Concurrency Control).
- **Multi-Collection Transactions**: Supports ACID transactions across multiple collections using **JavaScript-based transaction blocks**.
- **AQL Query Transactions**: When using ArangoDB Query Language (AQL), ArangoDB **implicitly** manages transactions for operations.

2. Scalability & High Availability with Sharding & Replication

- **Sharding (Data Partitioning)**:
  - ArangoDB distributes data across **multiple shards**.
  - Uses a **consistent hashing mechanism** for load balancing.
  - SmartGraphs optimize sharding in graph databases.
- **Replication**:
  - Uses **Leader-Follower replication** for read scalability.
  - Followers asynchronously replicate data from the leader.
- **High Availability (HA)**:
  - **Active Failover**: If the leader node fails, a follower is **automatically promoted**.
  - **Cluster Mode**: Uses multiple coordinators to distribute requests, ensuring no single point of failure.

3. ACID Transactions & Data Integrity

- **Atomicity**: Transactions are either fully completed or fully rolled back.
- **Consistency**: Ensures constraints (like unique indexes) remain intact.
- **Isolation**: Uses **snapshot isolation** with MVCC to avoid dirty reads.
- **Durability**: Data is stored in **Write-Ahead Logs (WAL)**, ensuring recovery after crashes.

Bibliography

1. ArangoDB Documentation – https://www.arangodb.com/docs/stable/
2. ArangoDB Transactions – https://www.arangodb.com/docs/stable/transactions.html
3. ArangoDB Replication – https://www.arangodb.com/docs/stable/replication.html
4. ArangoDB Sharding – https://www.arangodb.com/docs/stable/sharding.html

Security in ArangoDB

Security is important in databases because it helps **protect data from unauthorized access, modification, or loss**. ArangoDB provides several security features to keep data safe. Here are the key ones:

1. User Authentication

- **What it does:** Ensures that only authorized users can access the database.

- **How it works:** Users must provide a **username and password** to connect.
- **Example:** If you try to log in without the right password, ArangoDB **denies access**.

2. Role-Based Access Control (RBAC)

- **What it does:** Controls what actions users can perform.
- **How it works:**
    - Admins can assign **roles** to users (e.g., **read-only, editor, admin**).
    - A user with **read-only** access can **view data but not change it**.
    - A user with **admin** access can **modify settings and data**.
- **Example:** A junior developer might get "read-only" access, while a senior developer gets "write" access.

3. Encryption

- **What it does:** Protects data from hackers by converting it into a secret code.
- **Types of encryption in ArangoDB:**
    - **Data in Transit (TLS/SSL)**: Protects data while it moves between users and the database.
    - **Data at Rest**: Encrypts stored data to prevent unauthorized access.
- **Example:** If a hacker intercepts an **unencrypted** password, they can steal it. If it's **encrypted**, they only see scrambled data.

Why is this Important?

- Prevents **unauthorized access** to sensitive data.
- Ensures **only the right people** can make changes.
- Keeps data **safe from hackers**.

Summary

- **Authentication**: Users need a username & password.
- **RBAC**: Users get different permissions based on roles.

- **Encryption**: Protects data from hackers by scrambling it.

5. Tools for Managing ArangoDB:

- **Available Tools:**

Main Tools Available for ArangoDB

(*1) **REST API** stands for **Representational State Transfer Application Programming Interface**. It's a way for different software systems to communicate over the web using standard HTTP methods like GET, POST, PUT, DELETE, etc.

5

1. ArangoDB Web Interface (Browser-Based UI)

- **What it does**: A **graphical** tool to manage databases, run queries, and monitor performance.
- **Best for**: Beginners, administrators, and those who prefer a visual interface.
- **How to access it**: Open your browser and go to:
  - http://localhost:8529
- **Features**:
  - Run **AQL queries** (ArangoDB Query Language).
  - View **collections and documents**.
  - Manage **users and permissions**.
  - Monitor **database performance**.

Practice: Running an AQL Query in the Web Interface

- Open http://localhost:8529 in your browser.
- Go to the **Query Editor**.
- Run this query to fetch all documents from a collection:

```
FOR doc IN my_collection
  RETURN doc
```

ArangoShell (Command-Line Interface)

- **What it does**: A CLI tool for **scripting and executing commands**.
- **Best for**: Developers, DevOps engineers, and advanced users.
- **How to open it**: Run the following in a terminal: arangosh
- **Features**:
    - Run **AQL queries**.
    - Create and manage **databases** and **collections**.
    - Execute **batch scripts** for automation.

Practice: Creating a Collection in ArangoShell

```
// Start ArangoShell and connect to the database

db._createDatabase("testDB");


// Switch to the new database

db._useDatabase("testDB");


// Create a new collection

db._create("my_collection");

print("Collection created successfully!");
```

ArangoBackup (Backup and Restore Tool)

- **What it does**: Helps create backups of your ArangoDB database and restore them when needed.
- **Best for**: System administrators and anyone managing database safety.
- **How to use it**: Run commands in the terminal.

Practice: Creating a Backup

```
arangobackup create --output-directory /backup
```

This saves the database backup in /backup.

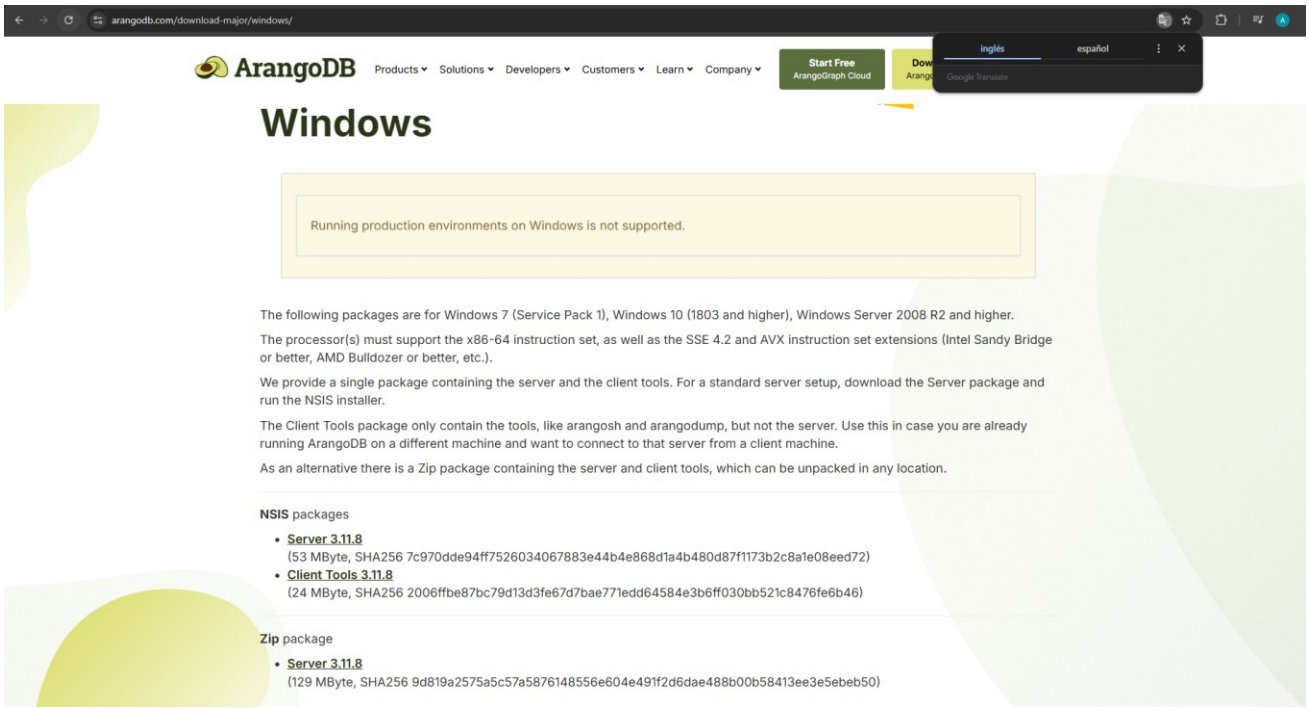
Restoring a Backup

arangobackup restore --input-directory /backup

Differences & Functions

Tool	Type	Best For	Functions
Web Interface	Graphical UI (Browser-based)	Beginners, Admins	Query execution, user management, performance monitoring
ArangoShell	Command-Line (CLI)	Developers, Advanced users	Scripting, collection management, database administration
ArangoBackup	Backup Tool (CLI-based)	SysAdmins, Backup Management	Backup creation, data restoration, disaster recovery

INSTALLING ARANGODB COMMUNITY



We go to the following route and download the zip package

https://arangodb.com/download-major/windows

We will then unzip it and go to our CMD and write:

Arangodb

And in a separate CMD:

Arangod

We then go to our browser and search **LocalHost:8529**, this will take us to the Login page of arangodb community.

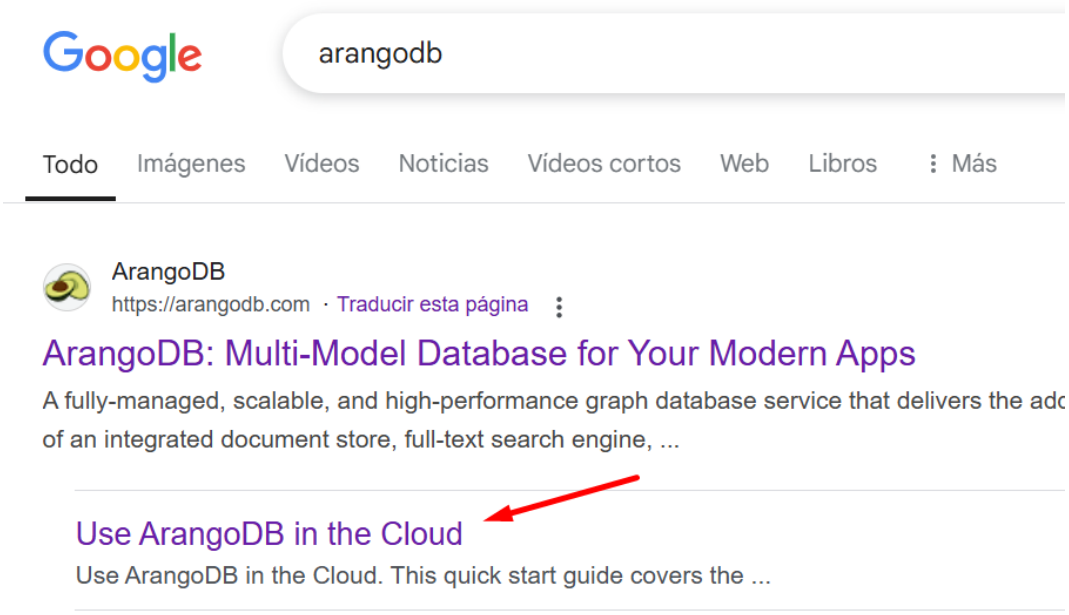
Log as root and no password and we are in.

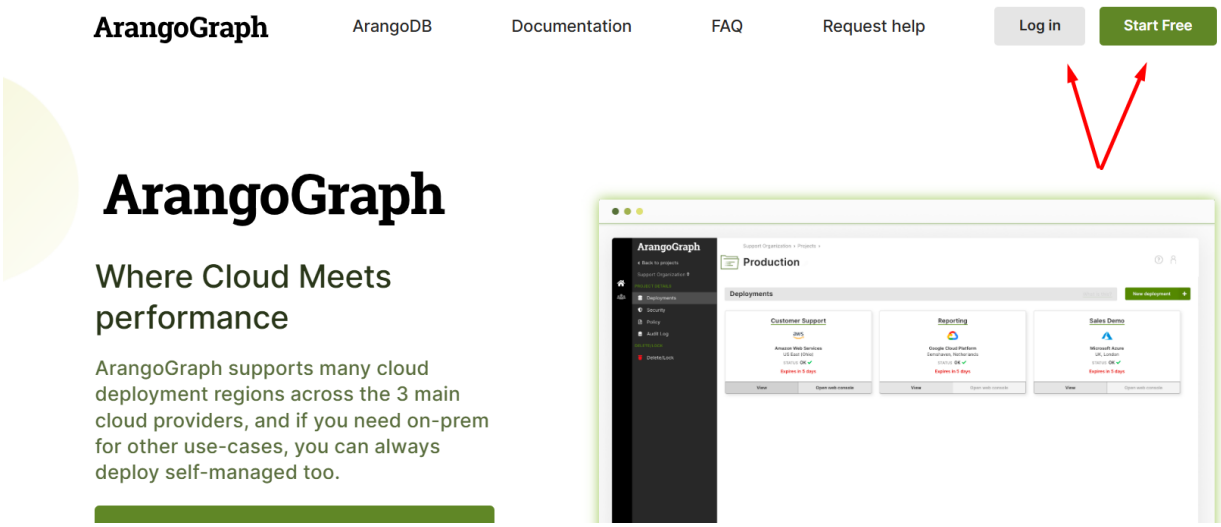
HOW TO GET INTO CLOUD:

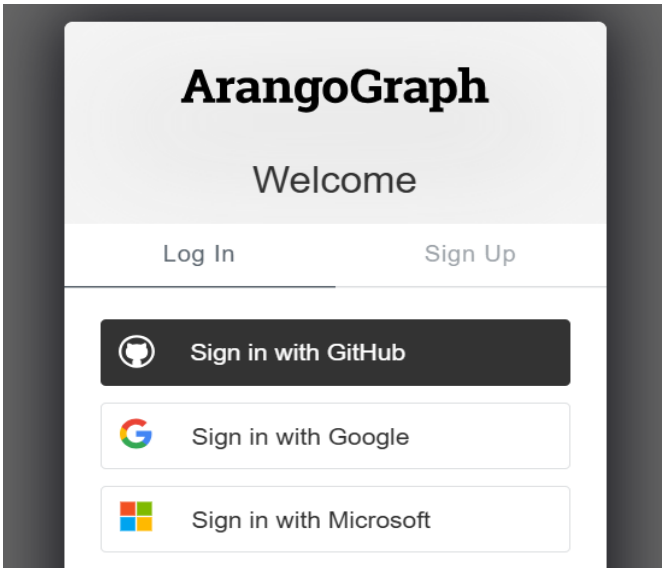
Search in Google > use arango cloud

Whether you have an account or not, you can press either of the two buttons.

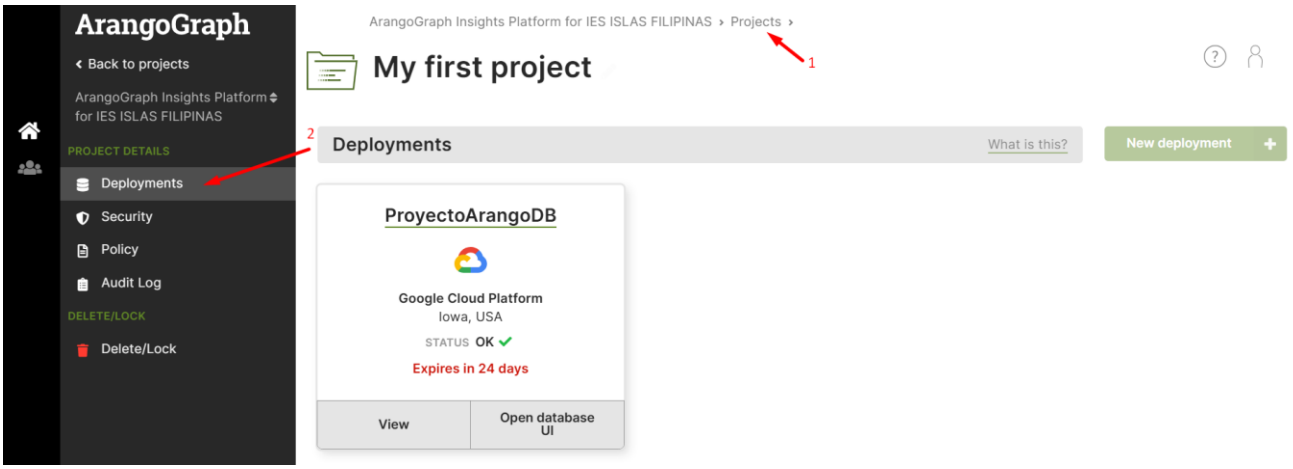
You will log in or sign up:

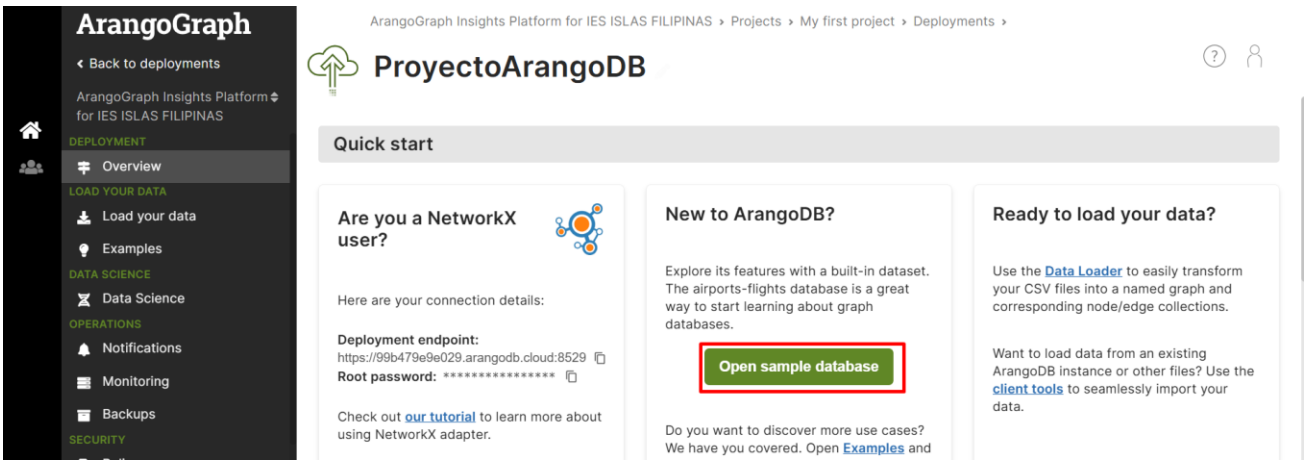






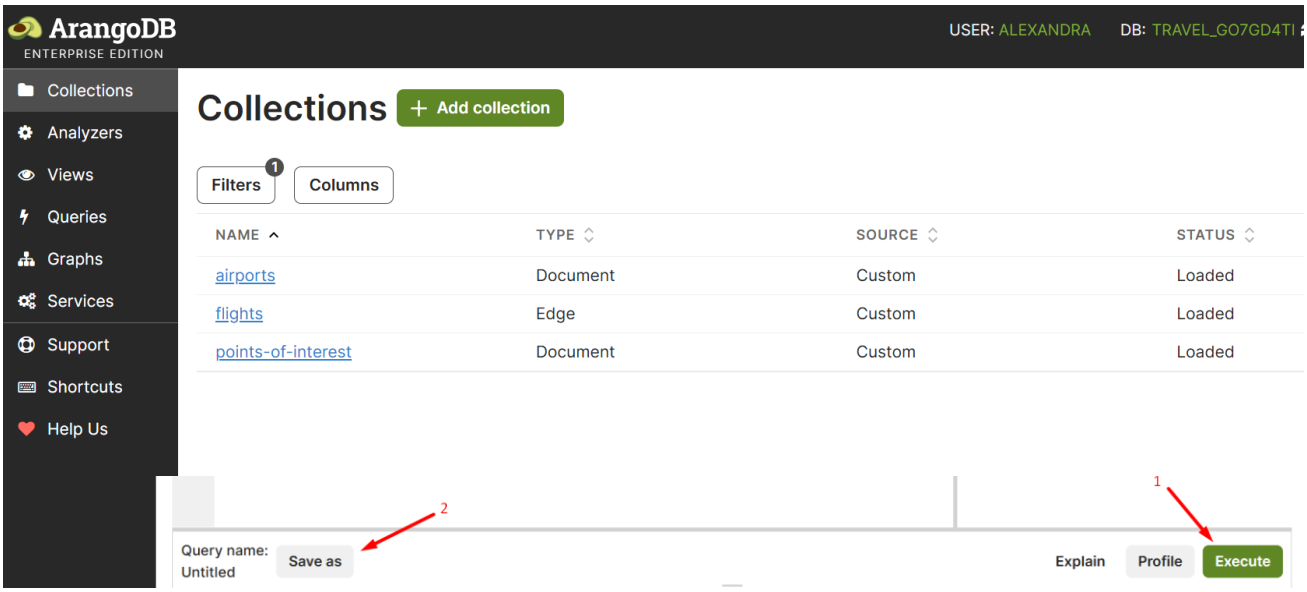
Then we got to Projects and you do your first deployments





You will have the name of your project there:

THIS IS YOUR DATABASE:



QUERIES:

CRUD

· **Create**: Use INSERT (AQL) or POST (HTTP API).

· **Read**: Use FOR and RETURN (AQL) or GET (HTTP API).

· **Update**: Use UPDATE (AQL) or PATCH (HTTP API).

· **Delete**: Use REMOVE (AQL) or DELETE (HTTP API).

1. Create (C)

- **AQL**: Use INSERT to add documents to a collection.

  Copiar
  INSERT { "_key": "123", "name": "John", "age": 29 } INTO users

- **HTTP API**: Use POST to create a document.

  Copiar
  curl -X POST "http://localhost:8529/_api/document/users" -H "Content-Type: application/json" -d '{"name": "John", "age": 29}'

---

2. Read (R)

- **AQL**: Use FOR and RETURN to retrieve documents.

  Copiar
  ```
  FOR user IN users
    RETURN user
  ```

- **HTTP API**: Use GET to retrieve a document by key.

  Copiar
  ```
  curl -X GET "http://localhost:8529/_api/document/users/123"
  ```

---

3. Update (U)

- **AQL**: Use UPDATE to modify a document's attributes.

  Copiar
  ```
  UPDATE "123" WITH { "age": 30 } IN users
  ```

- **HTTP API**: Use PATCH to update a document.

  Copiar
  ```
  curl -X PATCH "http://localhost:8529/_api/document/users/123" -H "Content-Type: application/json" -d '{"age": 30}'
  ```

---

4. Delete (D)

- **AQL**: Use REMOVE to delete a document.

  Copiar
  ```
  REMOVE "123" IN users
  ```

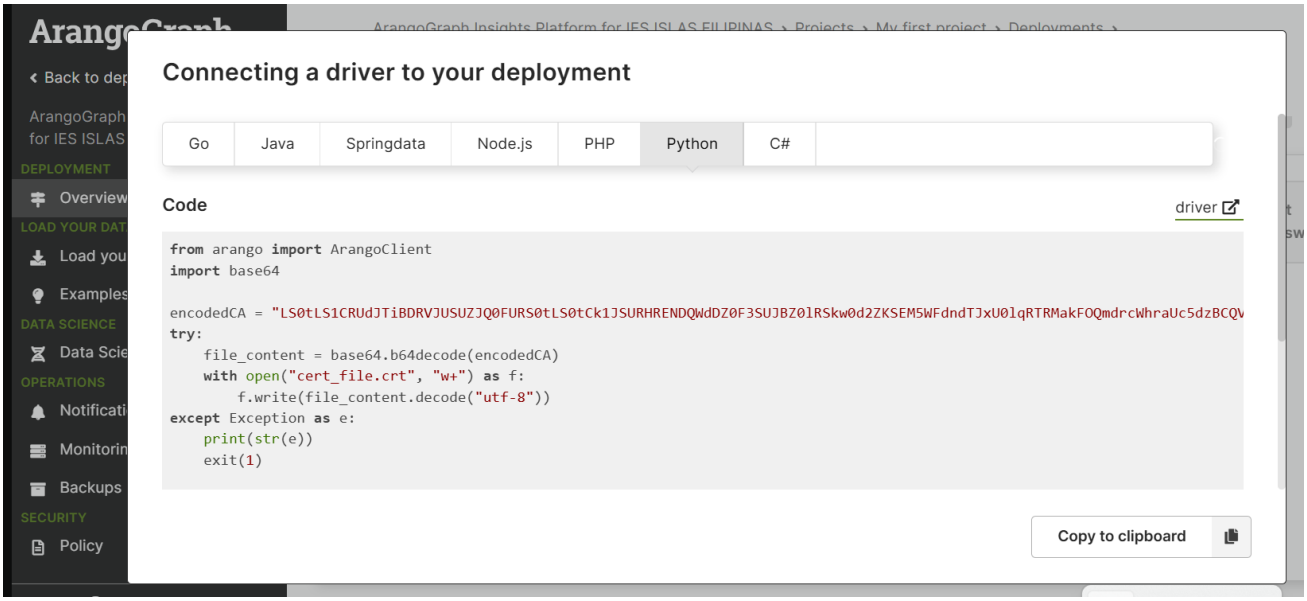
- **HTTP API**: Use DELETE to remove a document.

  Copiar
  ```
  curl -X DELETE "http://localhost:8529/_api/document/users/123"
  ```

Query 1000 elements 2.03 ms JSON Table

_key	_id	_rev	name	city	state	country	lat
00M	airports/00M	_jdVfnXO---	Thigpen	Bay Springs	MS	USA	31
00R	airports/00R	_jdVfnXO-_-	Livingston Municipal	Livingston	TX	USA	30
00V	airports/00V	_jdVfnXO--A	Meadow Lake	Colorado Springs	CO	USA	38
01G	airports/01G	_jdVfnXO--B	Perry-Warsaw	Perry	NY	USA	42

HOW TO GET DRIVERS FOR EVERY PROGRAMMING LANGUAGE:



LINKS TO OUR MAIN SOURCES

https://arangodb.com

https://www.ibm.com/es-es/think/topics/nosql-databases

https://www.geeksforgeeks.org/difference-between-relational-database-and-nosql/

https://www.dataversity.net/nosql-databases-advantages-and-disadvantages/

https://chatgpt.com/
