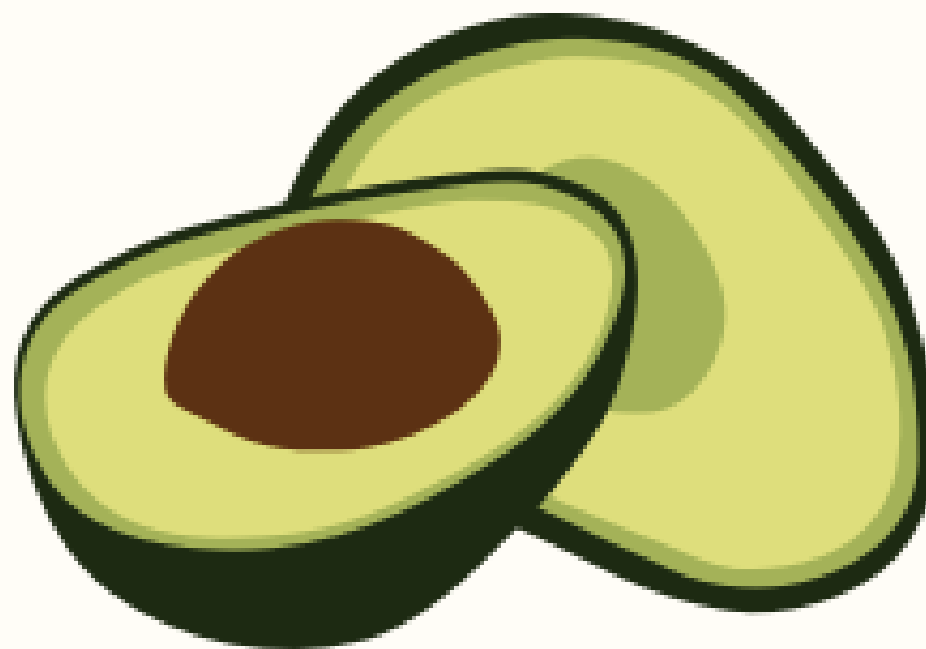




# ARANGO DB



**INTEGRANTES:**

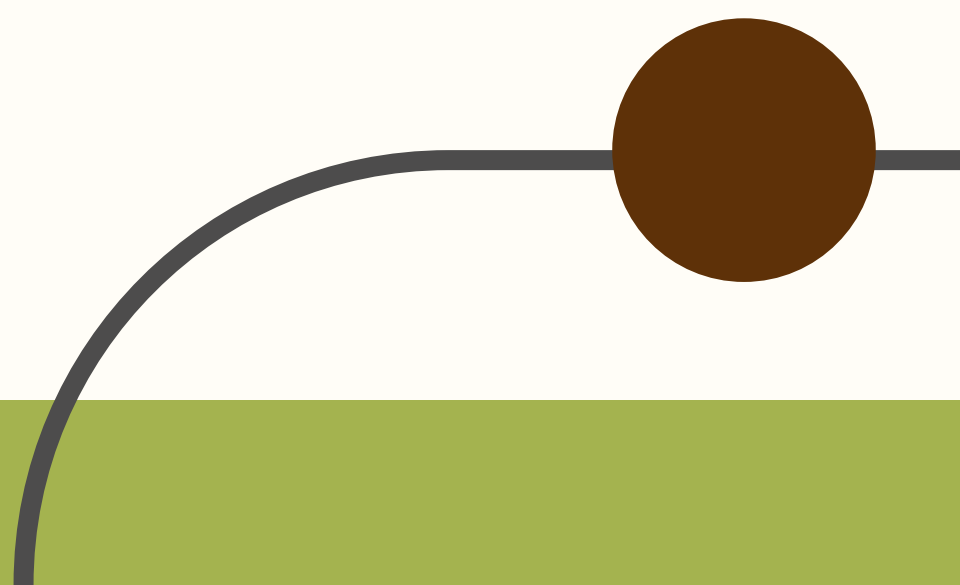
**MAXIMO SARNO, EMILY VOLTA, MARTÍN UBILLA**



# INTRODUCCIÓN

---

**ArangoDB se creó con el objetivo de simplificar la gestión de datos mediante la creación de una base de datos que soporte múltiples modelos de datos.**



# MODELO DE DATOS

---

**Es una base de datos NoSQL multimodelo, que combina documental, grafos y clave-valor. Su modelo de datos gira en torno a colecciones de documentos JSON los que se pueden conectar a aristas que permiten modelar grafos.**

# CARACTERÍSTICAS

---

- **Contiene un lenguaje AQL el que permite consultar datos como en SQL.**
- **Transacciones ACID completas, garantiza consistencia entre múltiples documentos.**
- **Drivers oficiales para múltiples lenguajes.**

# PROBLEMAS QUE RESUELVE

---

- Manejo de datos heterogéneos o semiestructurados y cambiantes.
- Evita la rigidez de los esquemas relacionales.
- Permite consultas complejas uniendo documentos y relaciones.
- Reduce la necesidad de múltiples sistemas.

# EJEMPLOS REALES

- **Actify:** Utilizaron arangoDB para modelar estructuras y ensambles de productos como grafos acíclicos dirigidos, lo que mejoró la gestión de datos
- **Cámara de comercio italiana:** ArangoDB superó consistentemente a otras, especialmente en consultas complejas de exploración de grafos.
- **Finite State:** con ArangoDB lograron una arquitectura optimizada, redujeron costos, escalaron de forma segura y mejoraron el rendimiento de las consultas.

# VENTAJAS

---

- Soporte ACID
- Lenguaje AQL Intuitivo
- Flexibilidad de esquemas

# DESVENTAJAS

---

- Ecosistema pequeño
- Consumo de recursos
- Rendimiento inferior en operaciones básicas
- AQL no soporta algunas operaciones DDL

## cuando USARLO

---

- Se necesita manejar documentos JSON complejos con estructura flexible.
- En aplicaciones que requieren consultas complejas.
- Se busca una sola base multimodelo.

## cuando NO USARLO

---

- Para aplicaciones muy simple que solo necesitan clave valor.
- En sistemas que demandan disponibilidad extrema sobre consistencia.



# COMPONENTES PRINCIPALES

---

**Agents:** Son el "cerebro" del cluster. Usan el algoritmo RAFT para mantener el consenso y la configuración de nodos.

**Coordinators:** Actúan como "puerta de entrada". Reciben las consultas (AQL, REST, GraphQL), las descomponen y las reparten entre los servidores de datos. También balancean la carga.

**DB Servers:** Almacenan los datos reales y ejecutan diferentes operaciones.

# TEOREMA CAP

---

- **Garantiza consistencia fuerte en operaciones críticas.**
- **Tolera particiones usando consenso Raft.**
- **Puede sacrificar disponibilidad en escenarios donde no hay mayoría de nodos para tomar decisiones (no permite escrituras “a ciegas” en todos los casos).**
- **ArangoDB prioriza la consistencia y la tolerancia a particiones (CP), sacrificando algo de disponibilidad en situaciones de fallos de red o caídas de líderes.**

# ESCALABILIDAD

---

- Escala horizontalmente con sharding y réplicas, ideal para grandes volúmenes de datos y alta concurrencia.
- También escala verticalmente, pero limitado al hardware del servidor.
- Requiere configuración de cluster (Agents, Coordinators, DB-Servers) para aprovechar la escalabilidad horizontal.
- ArangoDB fue diseñado para escalar horizontalmente en clusters distribuidos, aunque también puede escalar verticalmente, pero con menor beneficio.





# Instalación

# INSTALACIÓN

- 1.- Abrir Docker y CMD
- 2.- Ejecutar contenedor arangodb:

```
docker run -d -p 8529:8529 -e ARANGO_ROOT_PASSWORD=password123 --name arango-practica arangodb
```

- 3.- Abrir Docker e ingresar al Localhost

<input type="checkbox"/>	Name	Container ID	Im...	↓	Port(s)
<input type="checkbox"/>	 arango-practica	42fb84b07b3f	<a href="#">arangodb</a>		<a href="#">8529:8529</a> 

- 4.- Ingresa user: root y la contraseña escrita en el paso 2



# Implementación básica

# EJEMPLOS CRUD

- Primero debemos crear la base de datos y colecciones (2 opciones)
- Se crea un archivo “setup.js”

```
db = require('@arangodb').db; → Carga las herramientas de ArangoDB  
db._createDatabase('practica'); → Crea una base de datos nueva  
db._useDatabase('practica'); → Entra a usar esa base de datos  
db._create('usuarios'); → Crea una tabla/colección  
db._create('productos'); → Crea otra tabla/colección  
... etc  
db.usuarios.save([...]); → Guarda datos en la tabla  
db.productos.save([...]);  
... etc
```

- **En este ejemplo hay 4 colecciones: Usuarios, Productos, Pedidos, Cursos. En donde Pedidos relaciona a Usuarios y Productos.**

```
db = require('@arangodb').db;

// Crear base de datos
db._createDatabase('practica');
db._useDatabase('practica');

// Crear colecciones
db._create('usuarios');
db._create('productos');
db._create('pedidos');
db._create('cursos');

// Insertar datos de ejemplo en usuarios
db.usuarios.save([
  { nombre: "Ana", edad: 25, ciudad: "Madrid", email: "ana@email.com", activo: true, puntos: 150 },
  { nombre: "Carlos", edad: 32, ciudad: "Barcelona", email: "carlos@email.com", activo: true, puntos: 75 },
  { nombre: "Elena", edad: 28, ciudad: "Valencia", email: "elena@email.com", activo: false, puntos: 200 },
  { nombre: "David", edad: 35, ciudad: "Madrid", email: "david@email.com", activo: true, puntos: 300 },
  { nombre: "Sofia", edad: 22, ciudad: "Sevilla", email: "sofia@email.com", activo: true, puntos: 50 },
  { nombre: "Miguel", edad: 29, ciudad: "Bilbao", email: "miguel@email.com", activo: false, puntos: 125 }
]);
```



- 
- **Abrimos el CMD**
  - **Copiamos el archivo al contenedor**

```
docker cp setup.js arango-practica:/setup.js
```

- **Ejecutamos el script**

```
docker exec -it arango-practica arangosh --  
server.password password123 --javascript.execute  
/setup.js
```

Para el ejemplo de lectura tenemos la siguiente query:

```
FOR u IN usuarios
  FILTER u.nombre == "Ana"
  RETURN u
```

Query 1 element 0.569 ms

```
1 [
2 {
3   "_key": "355",
4   "_id": "usuarios/355",
5   "_rev": "_kRQyWya---",
6   "nombre": "Ana",
7   "edad": 25,
8   "ciudad": "Madrid",
9   "email": "ana@email.com",
10  "activo": true,
11  "puntos": 150
12 }
13 ]
```

Para el ejemplo de update tenemos la siguiente query:

```
FOR u IN usuarios
  FILTER u.nombre == "Carlos"
  UPDATE u WITH { puntos: 120 } IN usuarios
```

Query 1 element 0.226 ms

nombre	puntos
Carlos	120

Para el ejemplo de delete tenemos la siguiente query:

**FOR u IN usuarios**

**FILTER u.nombre == "Miguel"**

**REMOVE u IN usuarios**

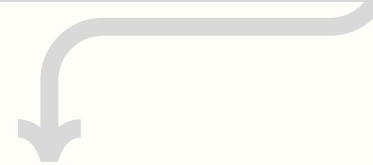
nombre
Ana
Elena
David
Sofia
Carlos

**Buscar usuario(s) activo(s) con mas de 100 puntos:**

**FOR u IN usuarios**

**FILTER u.activo == true AND u.puntos > 100**

**RETURN {nombre :u.nombre, puntos: u.puntos}**



nombre	puntos
Ana	150
David	300
Carlos	120

## Ejemplo de inner join

```
FOR u IN usuarios
```

```
  FOR p IN pedidos
```

```
    FILTER u.nombre == p.usuario
```

```
    RETURN {
```

```
      usuario: u.nombre,
```

```
      pedido: p.id_pedido
```

```
    }
```

usuario	pedido
Ana	P001
Carlos	P002
Elena	P003
Ana	P004
David	P005

## Usuarios con cantidad de pedidos

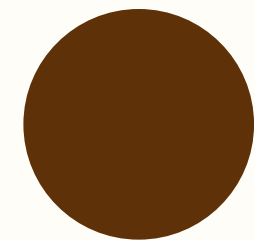
**FOR p IN pedidos**

**COLLECT usuario = p.usuario WITH COUNT INTO  
cantidad**

**RETURN { usuario, cantidad\_pedidos: cantidad }**



usuario	cantidad_pedidos
Ana	2
Carlos	1
David	1
Elena	1



Gracias por  
su atención

