Bases de Datos III

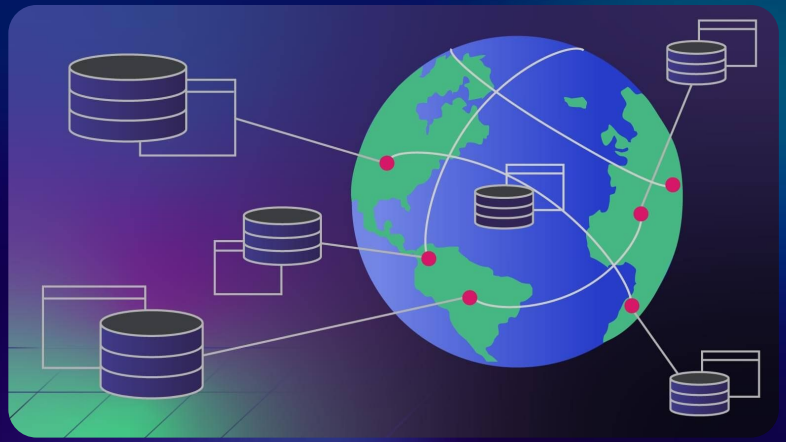
Sistema Distribuido Simulado

Becerra Ignacio
Goñe Juan Ignacio
Ravenna Sofia
Requena Melina

Objetivo

Diseñar e implementar un sistema distribuido simulado que:

- garantice **consistencia causal** entre operaciones de lectura y escritura
- utilice **vectores de versión** (vector clocks)
- utilice **propagación controlada** de operaciones entre nodos.



Breves Definiciones

Consistencia Causal

Vectores de Version

Propagación Controlada

Consistencia Causal

Si A pasa luego de B, entonces A debe ejecutarse siempre luego de B en todas las replicas.

Vectores de Version

ED que registran historial de actualizaciones en un sistema distribuido para comparar orden causal entre eventos.

Propagación Controlada

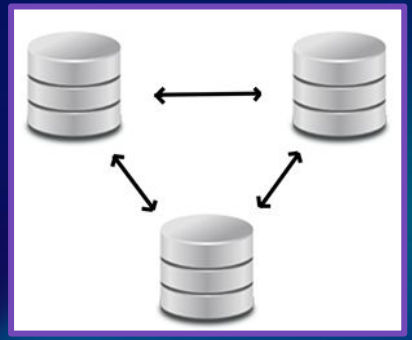
Estrategia para decidir cuándo y cómo cada nodo envía sus operaciones a los demás.
En un sistema con CC, no se aplica una operación hasta que sus causas hayan llegado.
Por eso, cada nodo tiene una **cola de espera** donde guarda operaciones que todavía no puede aplicar.

Arquitectura de la Solución

3 procesos locales expuestos por HTTP:

- N1 en puerto 8001
- N2 en puerto 8002
- N3 en puerto 8003

```
{
  "dni": "43505984",
  "nombre": "Meli Requena",
  "carrera": "Ingeniería Informática",
  "anio": 4,
  "nota_promedio": 8.5
}
```



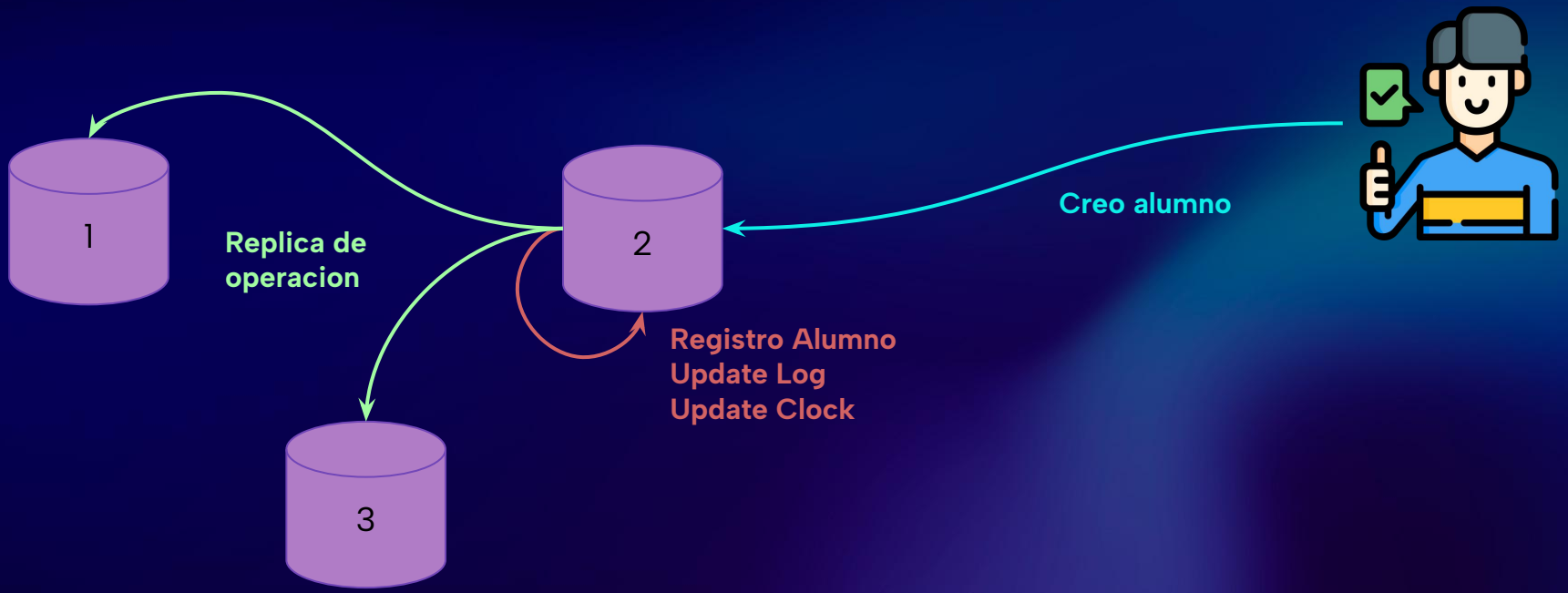
Cada nodo con:

- BDD dictionary local (**store** = {} con claves y valores).
- Un vector clock (**vc** = {"n1": 0, "n2": 0, "n3": 0}).
- **Cola de espera** para operaciones que aún no se pueden aplicar.
- Un log JSON para registrar operaciones.

Endpoints:

- PUT
- GET
- POST

Arquitectura de la Solución



Stack de Tecnologias Aplicadas

FastAPI → Python 3.11.09

- Framework web para Python
- Crea endpoints HTTP
- Cada nodo es un pequeño servidor que:
  - Atiende peticiones HTTP: Lee/escribe claves
  - Mantiene su vector de versión
  - Actualiza un log con operaciones
  - Propaga operaciones a otros nodos (si se cumple la causalidad)

01

Implementacion

Que es un Endpoint

Un endpoint es una dirección o ruta que posee el nodo para que clientes soliciten operaciones.

Cada endpoint tiene:

- Una URL (ej: *http://localhost:8001/kv/x*).
- Un método HTTP (GET, PUT, POST, etc.).
-
- Una función en tu código que decide qué hacer cuando alguien solicita ese endpoint.

Endpoints utilizados

GET /alumnos
POST /alumnos
GET /alumnos/{dni}
PUT /alumnos/{dni}

POST /replicate
GET /health
GET /log
GET /queue

GET /alumnos

Devuelve todos los alumnos guardados en este nodo.

POST /alumnos

Crea un alumno localmente y actualiza el vector clock del nodo. Luego replica la operación a los demás nodos.

```
{
  "dni": "string",
  "nombre": "string",
  "carrera": "string",
  "anio": 0,
  "nota_promedio": 0
}
```

Endpoints utilizados

GET /alumnos
POST /alumnos
GET /alumnos/{dni}
PUT /alumnos/{dni}

POST /replicate
GET /health
GET /log
GET /queue

GET /alumnos/{dni}

Devuelve un alumno específico por DNI. Si no existe, retorna rta 404.

PUT /alumnos/{dni}

Actualiza los datos de un alumno localmente y propaga el cambio. Incrementa el vector clock y replica la operación.

```
{
  "dni": "string",
  "nombre": "string",
  "carrera": "string",
  "anio": 0,
  "nota_promedio": 0
}
```

Endpoints utilizados

GET /alumnos
POST /alumnos
GET /alumnos/{dni}
PUT /alumnos/{dni}

POST /replicate
GET /health
GET /log
GET /queue

POST /replicate

Recibe una operación replicada desde otro nodo. Verifica si puede aplicarse según el vector clock. Si no, la guarda en la hold-back queue.

GET /health

Endpoint de chequeo: nos muestra el id del nodo y el VC.

GET /log

Devuelve el log local completo de este nodo. Permite ver las operaciones realizadas o recibidas.

GET /queue

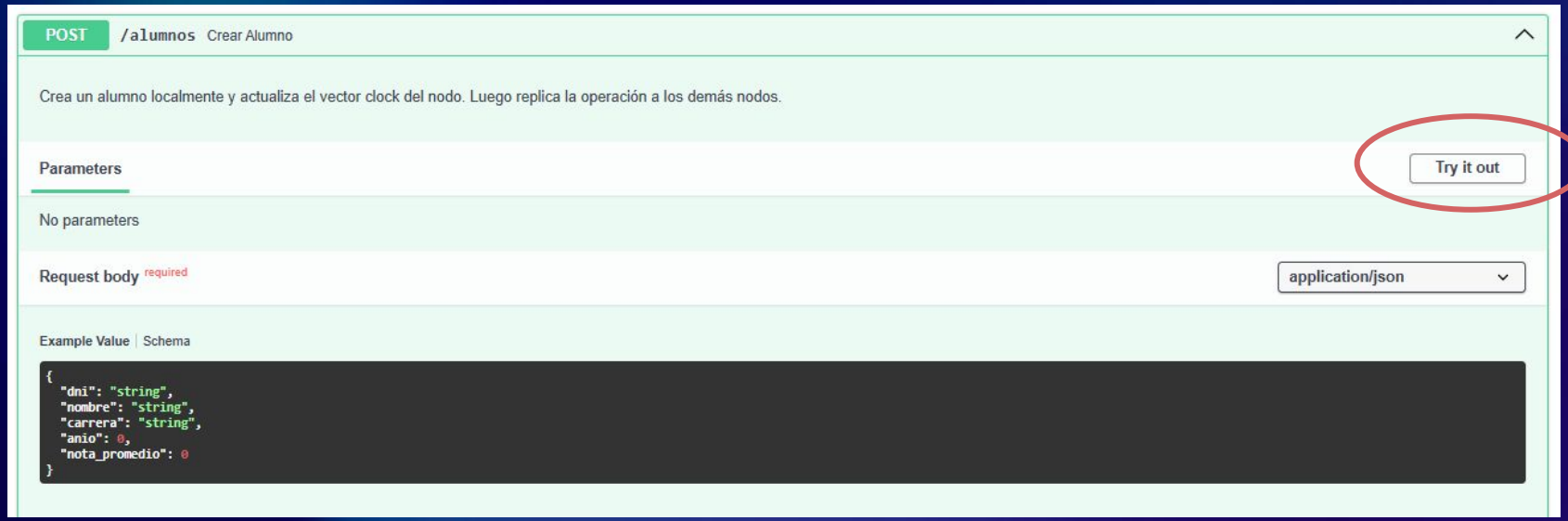
Devuelve las operaciones en espera.

Nodo n3 0.1.0 OAS 3.1

/openapi.json

default

- GET `/health` Health
- GET `/alumnos` Listar Alumnos
- POST `/alumnos` Crear Alumno
- GET `/alumnos/{dni}` Obtener Alumno
- PUT `/alumnos/{dni}` Actualizar Alumno
- POST `/replicate` Recibir Replicacion
- GET `/log` Ver Log
- GET `/queue` Ver Hold Back Queue


POST
/alumnos
Crear Alumno
Crea un alumno localmente y actualiza el vector clock del nodo. Luego replica la operación a los demás nodos.
Parameters
Try it out
No parameters
Request body
required
application/json
Example Value
Schema
{
"dni": "string",
"nombre": "string",
"carrera": "string",
"anio": 0,
"nota_promedio": 0
}

Gracias!

Preguntas?