Reporte de Evaluación de las Tecnologías Documentales

Recurso crucial: https://github.com/MattiasMR/EV01_advdb

Contextualización

Se nos pidió realizar una implementación y una evaluación de tres tecnologías diferentes. DynamoDB, MongoDB y Apache Cassandra. Para lograr esto se nos pidió generará APIS para realizar cada una de las actividades solicitadas en esta evaluación. Las APIS tenían que ser implementadas en NodeJS. El entregable de esta actividad consiste en:

- Archivos de creación de las bases de datos
- Archivos de creación o códigos fuentes de las APIS
- Archivos de poblamiento de las bases de datos
- Reporte de evaluación de las tecnologías documentales que abarca esta evaluación La implementación específica tiene que ser para una base de datos para una veterinaria.

Cómo abordé el problema y mi proceso de pensamiento al escoger un motor

Antes de siquiera pensar en algún motor de base de datos, lo primero que hice fue leer el problema, definir qué necesidades tenía que cumplir mi base de datos (qué información iba a tener que tener para poder mostrar los datos que me pedían). Luego hice un checklist de todo eso mientras iba modelando tablas con atributos. Una vez visto el problema y desglosado, identifique una tabla paciente, médico con su lógica específica de desactivación, tutores, la necesidad de una ficha médica y un historial de procedimientos y consultas. Ya había probado un poco esto considerando que en clase se nos había entregado como actividad crear la misma base de datos, más básica eso sí, y poblar, por lo tanto ya lo había probado con uno de los motores, DynamoDB.

Como ya tenía un poco de experiencia decidí partir con DynamoDB y luego replicar todo en MongoDB y en Cassandra. Al principio me costó recordar bien cómo era el framework de serverless que se nos había presentado de dynamo pero una vez le agarré la mano, mediante harto tiempo de investigación y documentación, fue muy fácil, encontré un tutorial en la misma página de serverless donde mostraban ejemplos y un repositorio de github con muchísimos ejemplos de implementaciones y ya de ahí fue trabajar con node y seguir el checklist que ya había trabajado. Si le dediqué un poco más de tiempo al inicio que el resto ya que había que retocar cosas del modelo como recién lo estaba probando, y las queries tuve que notar el límite que impone para los índices y se notó bastante en los tiempos para las querys.

Después de haber trabajado todo en DynamoDB, y asegurarme que tenía todo lo que se pedía, ahí recién me fui por MongoDB, donde al inicio estaba muy muy perdido, no sabía como empezar, si utilizar Atlas, Serverless o directo en una instancia EC2. Fue donde le dediqué la mayor cantidad de tiempo a investigar y leer documentaciones, y probé primero en atlas y luego me pasé a EC2 ya que me sentía más cómodo y además era bien transparente con mucho de los costos y configuraciones.

Una de las dificultades grandes que me detuvo harto tiempo fue la implementación de las funciones, yo intenté de migrar todo y simplemente cambiar la configuración de la implementación (tablas, y modelos) y como se poblaba, pero lo que me detuvo por mucho tiempo fue la diferencia en cómo interactúan con el sistema de AWS, ya que serverless era mediante AWS Lambda y Mongo lo estaba tratando de hacer con express, que tiene un sistema distinto de comunicación, eventos a comparación de status: req-resp. Una vez pasada esa valla ya la implementación fue fácil, sobre todo porque para el control anterior investigué un poco como era la estructura de mongo, y su estilo de trabajar con documentos ya me era familiar.

Ahora el motor que en lo personal es el más difícil fue Cassandra. La similitud que tiene con SQL pero sin serlo se me hizo bien confusa. Aquí de inmediato me fuí por la alternativa de trabajar en una instancia en EC2 considerando sobre todo mi falta de experiencia, quería sacar el método de como utilizarlo de la ecuación. Crear un schema fue relativamente similar, eso sí en el proceso de investigación, encontré varias prácticas que se hacían, como crear tablas tipo, y cómo funcionaban los índices, y la replicación de los nodos que se me hacía bastante confuso, y traté de hacerlo lo más idéntico posible a como lo hice en mongo y dynamo lo cual fue un desafío.

Tuve varios problemas con los tipos de datos que especificaba dentro de las tablas por los UUID que generaba y los float y las listas con texto. Una vez pasado eso, pude poblarlo finalmente y empezar con las queries. Los select iniciales eran idénticos a sql, super familiar y nada complicado, luego manejar los tipos de datos era complicado, no existían los groupby en cassandra sql y los joins eran raros así que trate de tener todo lo más junto posible, sin perder la lógica, pero tuve que hacer “scans” dentro de otras querys. Tuve que notar que me hizo repensar la estructura de mi modelo varias veces.

También me tomé el tiempo de hacer un pequeño estudio de cómo sería una veterinaria, gracias a otro estudio que encontré saqué estas conclusiones. De este estudio del 2022 saqué que: "en Chile existen 12.482.679 perros y gatos con dueños" y "65% va al veterinario al menos una vez al año", quedando en 8.113.741 perros y gatos que van al veterinario al menos 1 vez al año. Por temas de conveniencia son 676.145 al mes y 3 consultas promedio por paciente (revisar, actualizarla, volver a revisar/actualizar) . Da un aproximado de 2 millones de consultas mensuales.

Para calcular los costos en base a esos números, utilice herramientas que entregaba el mismo AWS con una calculadora de precios que tiene y otros números que entrega para todas las configuraciones que aplique.

Toma de decisión

Mientras iba haciendo todo este proceso iba documentando todo en un bloc de notas. Haciendo paso a paso lo que hacía también me ayudó ahora para armar esta tabla comparativa que me ayudó a tomar la decisión de los tres motores donde 1-3 es Pobre, 4-7 es Bueno y 8-10 es Excelente.

Motor de BD / Factor	**DynamoDB**	**MongoDB**	**Cassandra**
Costo base de datos	$0.00065 por WCU y $0.00013 por RCU (1 Unidad de escritura hasta 1KB por segundo y 0.5 unidades de lectura hasta 4kb por segundo)	3000 IOPS gratis y 0,005 USD/IOPS provisionadas al mes	3000 IOPS gratis y 0,005 USD/IOPS provisionadas al mes

Motor de BD / Factor	DynamoDB	MongoDB	Cassandra
Costo plataforma física	0 para los primeros 25gb, luego $0.25/GB mensual para el almacenamiento	$0.0139 por hora para la instancia y $0.08 USD/GB al mes para el almacenamiento	$0.0499 por hora para la instancia y $0.08 USD/GB al mes para el almacenamiento
Costo Mensual Veterinaria	$2.87 USD (2.028 millones de solicitudes)	$10.08 USD 24/7(1 nodo)	$35.936 USD 24/7 (1 nodo)
Tiempos de respuesta	8 - En promedio 914ms por api request y 1m02s por deploy/poblar	10- En promedio 48.4 ms desde la misma instancia pero 247ms desde bruno localmente para las api y 0.694s para poblar	9 - En promedio 87.6 ms desde la misma instancia pero 248ms desde bruno localmente para las api y 8.335s para poblar
Facilidad de instalación	10 - Extremadamente fácil con los ejemplos del repositorio y serverless.yml	8 - Seguir un tutorial, dificil al tratar con Atlas	6 - Seguir un tutorial pero mucho ojo con las versiones y problemas de memoria con instancias pequeñas, cqlsh no me funcionó
Facilidad de uso	10 - Extremadamente fácil trabajar en local con serverless y era muy flexible con los modelos	9 - el driver de mongo para node fue muy fácil, atlas fue el unico inconveniente, muy flexible con los modelos	7 - no era necesariamente dificil, pero si muy distinto y limitante con los modelos en comparación a los otros dos
Disponibilidad de Documentación	10 - Infinita en AWS y repositorios y tutoriales de serverless	9 - Mucha documentación y blogs con tutoriales, dificil con Atlas sobretodo porque lo están remodelando la página	9 - Tutoriales cortos con quickstart para ayudar y blogs escritos
Impacto en la industria	Top 3 según mercado y top 15 según popularidad	Top 1 y top 5 según popularidad	Top 5 y top 11 según popularidad
Modelo de datos	Documentos JSON clave-valor	Documentos BSON	Filas y columnas, "wide-column"
Flexibilidad de datos	9	10	7

Motor de BD / Factor	DynamoDB	MongoDB	Cassandra
Escalabilidad	10 - Completamente automatico	8 - Requiere configuración de clusters	8 - Diseñado para ser escalable horizontalmente y para distribuirlo en clusters
Query	7 - Limitado para consultas poderosas por los indices	9 - muy flexible, busquedas textuales, recuperación de información muy buena	8.5 - Complicado por el tema de tablas de tipo, o indexaciones, pero si se sabe que hay que hacer definitivamente muy poderoso para evitar las limitaciones

Puntajes:

- Dynamo: 74/80
- Mongo: 71/80
- Cassandra: 61.5/80

Si tuviese que elegir ahora, uno de estos motores de base de datos para un proyecto a semi-ciegas, escogería mongodb. Digo a semi-ciegas ya que mongodb según todo lo que vi, es el mejor en términos de flexibilidad/poder al momento de realizar consultas y almacenar datos, con su flexibilidad enorme al momento de almacenar datos y su buen poder de indexación dentro del texto, contextual. Si fuese para un proyecto que requiere alta accesibilidad, Cassandra, disponibilidad y escalabilidad muy alta, su funcionalidad de clusters es muy robusta. Y si fuese para cualquier otro tipo de proyecto, sobre todo completamente a ciegas a futuro, dynamodb por la flexibilidad no solo de los datos sino de la infraestructura en sí, se gestiona de manera automática, tiene todo el ecosistema de AWS para integrar necesidades futuras y es escalable sin problema, aunque también, está ligado a AWS que no siempre puede ser lo mejor. Teniendo casi la misma flexibilidad en el modelo que mongo. Entonces, respondiendo la pregunta, me quedaría con dynamodb si necesitara algo escalable con harto tráfico, Cassandra si ya construyera una app masiva y mongodb en la mayoría de los casos donde no es ambos.

Motor de base de datos escogido: **MongoDB**

También quería destacar el framework de serverless, que siento que podría haber utilizado también en mongodb y en cassandra, al ser un framework agnóstico de lenguaje y haberlo probado más.

Cómo replicar mi entorno de trabajo

Además de documentar todo en mi github https://github.com/MattiasMR/EV01_advdb

Voy a mencionar aquí cómo replicar mi entorno y explicar la estructura de los archivos. Decidí estandarizar lo mayor posible los tres entornos distintos para que la explicación sea lo más simple posible. La mayor diferencia está entre dynamodb y mongo con cassandra ya que dynamo fue hecho con serverless, pero aparte de un par de archivos no difiere mucho.

Dentro de cada carpeta, mongodb/ - dynamo/ - cassandra/, van a estar las carpetas controllers/ - models/ - routes/ (está no en dynamo), donde controllers va a almacenar los archivos que manejan las llamadas a los puntos de las APIs, en models se van a encontrar los archivos de creación (tablas, poblar, colecciones, etc) y en routes las definiciones de los endpoints de las APIs, nuevamente en dynamo no se encuentra ni los archivos de creación, si el de poblar, ni las rutas, ya que todo eso es tratado en el serverless.yml.

También van a estar los archivos package.json con los requerimientos de cada entorno, un app.js para mongodb y cassandra donde se levanta el servidor con los endpoints y se maneja la conexión a la base de datos, y en dynamo se utiliza el serverless.yml que se encarga de todo eso.

Una vez entienden la estructura, para replicar hay que asegurarse de tener node 18 en cualquier entorno, seguir los pasos de instalación de cada uno (uno para mongo y otro para cassandra), y probar las apis. Está todo documentado en el github paso por paso para cada entorno, pero lo voy a incluir aquí también.

Requisitos Previos

> **Importante**: Asegúrate de tener Node.js versión 18.20.8 instalado. El proyecto no ha sido probado con otras versiones.

Configuración por Base de Datos

DynamoDB

Requisitos:

- Configurar credenciales AWS en `~/.aws/credentials`

Pasos de instalación:

1. **Clonar el proyecto**

```
git init
git remote add origin https://github.com/MattiasMR/EV01_advdb.git
git config core.sparseCheckout true
echo "dynamo/*" > .git/info/sparse-checkout
git pull origin main
cd dynamo/
```

2. **Instalar dependencias**

```
npm install
```

3. **Poblar las tablas**

```
node models/seed.js
```

4. **Desplegar el servidor**

```
sls deploy
```

5. **Probar las APIs**
    - Usar Postman, curl, o cualquier cliente REST

MongoDB

Infraestructura requerida:

- Instancia EC2: `t3.micro`
- OS: Ubuntu Server 24.04 LTS x64
- Almacenamiento: 1x8 GiB gp3

Pasos de instalación:

1. **Instalar MongoDB**

```
# Instalar mongodb-server-8.0
https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-
ubuntu/
```

2. **Clonar el proyecto**

```
git init
git remote add origin https://github.com/MattiasMR/EV01_advdb.git
git config core.sparseCheckout true
echo "mongodb/*" > .git/info/sparse-checkout
git pull origin main
cd mongodb/
```

3. **Instalar dependencias**

```
npm install
```

4. **Crear las colecciones**

```
node models/crearColecciones.js
```

5. **Poblar las tablas**

```
node models/seed.js
```

6. **Iniciar el servidor**

```
node app.js
```

7. **Obtener endpoints**
    - Copiar el API Gateway/URL que muestre la terminal
8. **Probar las APIs**

Apache Cassandra

Infraestructura requerida:

- Instancia EC2: `t2.medium`
- OS: Ubuntu Server 24.04 LTS x64
- Almacenamiento: 1x8 GiB gp3

Pasos de instalación:

1. **Instalar Cassandra**

```
# Instalar cassandra 4.1
https://medium.com/@agustinafassina_92108/install-cassandra-4-1-on-ubuntu-server-22-04-ec2-aws-15a4a730657e
```

2. **Clonar el proyecto**

```
git init
git remote add origin https://github.com/MattiasMR/EV01_advdb.git
git config core.sparseCheckout true
echo "cassandra/*" > .git/info/sparse-checkout
git pull origin main
cd cassandra/
```

3. **Instalar dependencias**

```
npm install
```

4. **Crear las tablas**

```
node models/crearTablas.js
```

5. **Poblar las tablas**

```
node models/seed.js
```

6. **Iniciar el servidor**

```
node app.js
```

7. **Probar las APIs**

Estructura de URLs

Cassandra y MongoDB (Express.js)

- **Base URL:** `http://localhost:3000/api/`
- **Formato:** `http://localhost:3000/api/[endpoint]`

DynamoDB (Serverless)

- **Base URL:** `https://[api-gateway-url]/`
- **Formato:** `https://[api-gateway-url]/[endpoint]`

Ejemplo de uso:

```
curl -X GET http://localhost:3000/api/paciente
```

Endpoints Disponibles

Pacientes

Método	Endpoint	Descripción
`POST`	`/paciente`	Crear paciente
`GET`	`/paciente`	Obtener todos los pacientes
`GET`	`/paciente/{id}`	Obtener paciente por ID
`PUT`	`/paciente/{id}`	Actualizar paciente

Médicos

Método	Endpoint	Descripción
POST	/medico	Crear médico
GET	/medico	Obtener todos los médicos
GET	/medico/{id}	Obtener médico por ID
PUT	/medico/{id}	Actualizar médico
PATCH	/medico/{id}/cambiarEstado	Cambiar estado del médico

Tutores

Método	Endpoint	Descripción
POST	/tutor	Crear tutor
GET	/tutor	Obtener todos los tutores
GET	/tutor/{id}	Obtener tutor por ID
PUT	/tutor/{id}	Actualizar tutor
GET	/tutor/{id}/pacientes	Obtener pacientes de un tutor

Búsquedas

Método	Endpoint	Descripción
GET	/paciente/{id}/historial	Historial del paciente
GET	/paciente/{id}/fichaClinica	Ficha clínica del paciente
GET	/paciente/{id}/vacunas	Vacunas del paciente
GET	/procedimientos/ranking	Ranking de procedimientos

Dashboard

Método	Endpoint	Descripción
GET	/dashboard/volumenGastoMensual	Volumen y gasto promedio por mes
GET	/dashboard/distribucionEspecialidades	Distribución por especialidad
GET	/dashboard/topMedicamentos	Top medicamentos
GET	/dashboard/evolucionIngresoVsCostes	Evolución ingresos vs costes
GET	/dashboard/demandaVacunasMensual	Demanda de vacunas por mes

Referencias

Todas las referencias usadas sin orden específico:
https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/on-demand-capacity-mode.html

https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/capacity-mode.html

https://aws.amazon.com/blogs/database/single-table-vs-multi-table-design-in-amazon-dynamodb/

https://www.mongodb.com/blog/post/quick-start-nodejs-mongodb-how-to-get-connected-to-your-database

https://www.mongodb.com/developer/languages/javascript/mongoose-versus-nodejs-driver/

https://docs.datastax.com/en/cql-oss/3.3/cql/cql_reference/cqlCreateKeyspace.html

https://docs.datastax.com/en/cql-oss/3.x/cql/cql_using/useCreateUDT.html

https://medium.com/@agustinafassina_92108/install-cassandra-4-1-on-ubuntu-server-22-04-ec2-aws-15a4a730657e

https://medium.com/@ibrahimhz/creating-your-first-backend-with-node-js-step-by-step-guide-892769af4cb0

https://6sense.com/tech/nosql-databases/mongodb-market-share#free-plan-signup

https://6sense.com/tech/nosql-databases/apache-cassandra-market-share#free-plan-signup

https://6sense.com/tech/nosql-databases/amazon-dynamodb-market-share

https://db-engines.com/en/ranking

https://www.gob.cl/noticias/primer-estudio-de-poblacion-animal-en-chile-revela-que-hay-12-millones-de-perros-y-gatos-con-duenos-y-4-millones-sin-supervision/

https://aws.amazon.com/dynamodb/pricing/provisioned/?nc1=h_ls

https://aws.amazon.com/es/dynamodb/pricing/on-demand

https://aws.amazon.com/es/blogs/database/new-amazon-dynamodb-lowers-pricing-for-on-demand-throughput-and-global-tables

https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-ubuntu/

https://www.serverless.com/framework/docs/getting-started