NestJSでつくるマルチテナントSaaS

Agenda

- はじめに
- NestJS × マルチテナント × MongoDB
- NestJS × マルチテナント × 認証
- NestJS × マルチテナント × カスタム要件
- NestJS × マルチテナント × 運用

NestJS × マルチテナント × MongoDB

TL;DR

- MongoDBのDatabaseでテナントを分割した
- ORMにMongooseを選定した
- MongooseのコネクションはDatabaseと1:1
- リクエストスコープでMongooseをInjectするとメモリ不足になる
- Serviceのメソッド実行時、適切なコネクションでModelを生成する

MongoDBのDatabaseでテナントを分割した

前提

- AWS DocumentDBを用いる
  - MongoDB互換のマネージドサービス
- テナントごとにデータを分離する必要がある

MongoDBによるマルチテナント構成

Databaseでテナントを分割した。

単位	Pros	Cons
Cluster	セキュリティが最も高い	インフラ費用、管理コストいずれも高い, テナント数に比例してコストが増加
Database	インフラ費用がテナント数に比例しない, RBACを活用しやすい	DatabaseをまたいだJOINのような処理ができないため、マスターデータとテナント固有データのJOIN処理は工夫が必要
	インフラ費用がテナント数に比	特定のテナントのCollectionのみアク

ORMにMongooseを選定した

MongoDB事例の多さから、手堅くMongooseを選定しました。

	NestJSサポート	Pros	Cons
mongoose	公式Moduleあり	実績が多い, NestJS公式ドキュメントでも取り上げられている, Transactionが使える	公式Moduleはあるが、複数Connectionをサポートしていない

リクエストスコープでMongooseをInjectするとメモリ不足になる

（意訳）MongoDBとのコネクションをリクエストスコープごとに生成すれば、リクエストごとに適切なテナントに接続できるよ。

- node.js - How to change a Database connection dynamically with Request Scope Providers in Nestjs? - Stack Overflow

リクエストスコープでMongooseをInjectするとメモリ不足になる

デモ:

```
# https://github.com/xhiroga/nestjs-meetup-online1-demo
% yarn dev
% curl localhost:33000/mytenant/cats
```

Serviceのメソッド実行時、適切なコネクションでModelを生成する

2通りのやり方が存在する。

1. DIを使わない。ConnectionのPoolを自前で持ち、サービスの呼び出し時にModelを生成する。
2. DIを使う。Modelをリクエストスコープで宣言し、ConnectionのPoolをするProviderをInjectする。

る

DIを使わないサンプル

```
@Injectable()
export const CatsService {
    constructor() {
        private readonly connectionProvider: ConnectionProvider;
    }

    async  getCats() {
        const connection = await this.connectionProvider.getConnection();
        const cats = await connectionProvider.model('cats').find();
        return cats;
    }
}
```

```
@Injectable()
export class ConnectionProvider{
    // 省略
    getConnection() {
        const tenant = this.request.params.tenantId;
    }
}
```

Serviceのメソッド実行時、適切なコネクションでModelを生成する

DIを使うサンプル

デモ

EOS