**📦 Environment Config Files in NestJS**

These .env files store environment-specific values like:

* Database credentials
* API keys
* Port, host, domain
* Feature flags (enable/disable features)

**📁 Folder/File Structure**

root/

├── .env # Default fallback env

├── .env.development # For development-specific values

├── .env.production # For production deployment

├── .env.test # For unit/integration test env

├── src/

│ └── config/

│ └── configuration.ts # Central config loader (type-safe)

│ └── validation.ts # Joi schema validation

**🔧 Step 1: Create .env Files**

**✅ .env (Fallback / shared values)**

APP\_NAME=MyNestApp

PORT=3000

NODE\_ENV=development

**✅ .env.development**

NODE\_ENV=development

DB\_HOST=localhost

DB\_PORT=5432

DB\_USERNAME=dev\_user

DB\_PASSWORD=dev\_pass

DB\_NAME=dev\_db

REDIS\_HOST=localhost

REDIS\_PORT=6379

**✅ .env.production**

NODE\_ENV=production

DB\_HOST=prod-db-host

DB\_PORT=5432

DB\_USERNAME=prod\_user

DB\_PASSWORD=prod\_secure\_password

DB\_NAME=prod\_db

REDIS\_HOST=prod-redis-host

REDIS\_PORT=6379

**✅ .env.test**

NODE\_ENV=test

DB\_HOST=localhost

DB\_PORT=5432

DB\_USERNAME=test\_user

DB\_PASSWORD=test\_pass

DB\_NAME=test\_db

**🔧 Step 2: Configure NestJS to Load .env**

Install dependencies:

npm i @nestjs/config joi

**✅ src/config/configuration.ts**

// src/config/configuration.ts

export default () => ({

appName: process.env.APP\_NAME,

port: parseInt(process.env.PORT, 10) || 3000,

nodeEnv: process.env.NODE\_ENV,

database: {

host: process.env.DB\_HOST,

port: parseInt(process.env.DB\_PORT, 10),

username: process.env.DB\_USERNAME,

password: process.env.DB\_PASSWORD,

name: process.env.DB\_NAME,

},

redis: {

host: process.env.REDIS\_HOST,

port: parseInt(process.env.REDIS\_PORT, 10),

},

});

**✅ src/config/validation.ts**

// src/config/validation.ts

import \* as Joi from 'joi';

export const validationSchema = Joi.object({

APP\_NAME: Joi.string().default('MyNestApp'),

PORT: Joi.number().default(3000),

NODE\_ENV: Joi.string().valid('development', 'production', 'test').required(),

DB\_HOST: Joi.string().required(),

DB\_PORT: Joi.number().default(5432),

DB\_USERNAME: Joi.string().required(),

DB\_PASSWORD: Joi.string().required(),

DB\_NAME: Joi.string().required(),

REDIS\_HOST: Joi.string().required(),

REDIS\_PORT: Joi.number().default(6379),

});

**🧩 Step 3: Import in AppModule**

// src/app.module.ts

import { Module } from '@nestjs/common';

import { ConfigModule } from '@nestjs/config';

import configuration from './config/configuration';

import { validationSchema } from './config/validation';

@Module({

imports: [

ConfigModule.forRoot({

isGlobal: true,

envFilePath: [`.env.${process.env.NODE\_ENV}`, '.env'],

load: [configuration],

validationSchema,

}),

],

})

export class AppModule {}

**🎯 Step 4: Use Config in Your Services**

// Example: src/app.service.ts

import { Injectable } from '@nestjs/common';

import { ConfigService } from '@nestjs/config';

@Injectable()

export class AppService {

constructor(private config: ConfigService) {}

getDatabaseHost(): string {

return this.config.get<string>('database.host');

}

getPort(): number {

return this.config.get<number>('port');

}

}

**✅ Best Practices Summary**

| **🔧 Practice** | **✅ Description** |
| --- | --- |
| .env.\* per environment | Keeps secrets and values separate |
| validation.ts | Prevents runtime failures from bad/missing values |
| ConfigModule.forRoot() | Enables global, validated, and layered env loading |
| Typed access via ConfigService | Avoids typo errors and improves maintainability |
| Don’t commit .env\* | Always add .env\* to .gitignore |
| Use .env.production with Docker/K8s | Match file with deployment strategy |