**1) Start a Vault server (dev mode for learning)**

**Option A — Native (terminal 1)**

**vault server -dev -dev-root-token-id=root**

**You’ll see:**

**API Address: http://127.0.0.1:8200**

**Root Token: root**

**Then export env (new terminal 2):**

**# PowerShell**

**$env:VAULT\_ADDR="http://127.0.0.1:8200"**

**$env:VAULT\_TOKEN="root"**

**# bash/zsh**

**export VAULT\_ADDR=http://127.0.0.1:8200**

**export VAULT\_TOKEN=root**

**Option B — Docker**

**docker run --name vault -p 8200:8200 \**

**-e VAULT\_DEV\_ROOT\_TOKEN\_ID=root \**

**-e VAULT\_DEV\_LISTEN\_ADDRESS=0.0.0.0:8200 \**

**hashicorp/vault:latest**

**2) Enable KV v2 and put your secrets**

**vault secrets enable -path=kv kv-v2 # mount: kv/**

**vault kv put kv/app DB\_USER=demo DB\_PASS=P@ssw0rd DB\_HOST=localhost DB\_NAME=mydb**

**vault kv get kv/app**

**You can add anything you need (API keys, SMTP creds, etc.).**

**3) Create the NestJS project & install deps**

**npm i -g @nestjs/cli**

**nest new nest-vault-demo**

**cd nest-vault-demo**

**npm i node-vault @nestjs/config**

**4) Add Vault environment hints (dev)**

**Create .env in project root:**

**# Vault connectivity**

**VAULT\_ADDR=http://127.0.0.1:8200**

**VAULT\_TOKEN=root # for dev; remove in prod**

**VAULT\_KV\_MOUNT=kv**

**VAULT\_KV\_PATH=app**

**# Fallbacks if Vault is down (optional)**

**DB\_HOST=localhost**

**DB\_NAME=mydb**

**DB\_USER=localuser**

**DB\_PASS=localpass**

**In production, don’t commit tokens. You’ll use AppRole (shown later).**

**5) Fetch secrets from Vault before Nest starts**

**We’ll fetch from Vault in main.ts, set process.env, then create the Nest app. This ensures any module (TypeORM, Mongoose, etc.) reads final values.**

**src/vault.bootstrap.ts**

**import vault from 'node-vault';**

**export type VaultSecrets = Record<string, string>;**

**async function sleep(ms: number) {**

**return new Promise((r) => setTimeout(r, ms));**

**}**

**export async function loadVaultSecrets(): Promise<VaultSecrets> {**

**const endpoint = process.env.VAULT\_ADDR ?? 'http://127.0.0.1:8200';**

**const token = process.env.VAULT\_TOKEN; // dev token; leave undefined for AppRole**

**const mount = process.env.VAULT\_KV\_MOUNT ?? 'kv';**

**const path = process.env.VAULT\_KV\_PATH ?? 'app';**

**const client = vault({**

**apiVersion: 'v1',**

**endpoint,**

**token, // if token undefined, you can do approle login below**

**// namespace: '...', // Vault Enterprise (optional)**

**});**

**// Example: AppRole login in prod**

**// if (!token && process.env.VAULT\_ROLE\_ID && process.env.VAULT\_SECRET\_ID) {**

**// const { auth } = await client.approleLogin({**

**// role\_id: process.env.VAULT\_ROLE\_ID,**

**// secret\_id: process.env.VAULT\_SECRET\_ID,**

**// });**

**// client.token = auth?.client\_token;**

**// }**

**// Simple retry (Vault might come up slightly later in Docker)**

**const maxAttempts = 5;**

**for (let attempt = 1; attempt <= maxAttempts; attempt++) {**

**try {**

**// KV v2 read: /v1/<mount>/data/<path>**

**const res = await client.read(`/${mount}/data/${path}`);**

**const data = (res?.data as any)?.data ?? {};**

**return data;**

**} catch (err) {**

**if (attempt === maxAttempts) throw err;**

**await sleep(1000 \* attempt); // backoff**

**}**

**}**

**return {};**

**}**

**// Apply secrets into process.env without clobbering existing values unless needed**

**export function applySecretsToEnv(secrets: VaultSecrets, overwrite = true) {**

**for (const [k, v] of Object.entries(secrets)) {**

**if (overwrite || !process.env[k]) {**

**process.env[k] = String(v);**

**}**

**}**

**}**

**src/main.ts**

**import { NestFactory } from '@nestjs/core';**

**import { AppModule } from './app.module';**

**import { loadVaultSecrets, applySecretsToEnv } from './vault.bootstrap';**

**async function bootstrap() {**

**// 1) Load from .env first (so VAULT\_\* vars exist)**

**// If you use @nestjs/config, import it inside AppModule; here we just use process.env.**

**// Alternatively, load dotenv here:**

**// require('dotenv').config();**

**// 2) Get secrets from Vault**

**try {**

**const secrets = await loadVaultSecrets();**

**applySecretsToEnv(secrets, true);**

**// console.log('Vault secrets loaded:', Object.keys(secrets));**

**} catch (e) {**

**// Decide your policy: fail fast or continue with local .env defaults**

**console.error('Failed to load Vault secrets at boot:', e);**

**// process.exit(1); // uncomment to hard-fail**

**}**

**// 3) Start Nest (all modules will now see final env values)**

**const app = await NestFactory.create(AppModule);**

**await app.listen(3000);**

**}**

**bootstrap();**

**6) Use the secrets anywhere (ConfigService-friendly)**

**Install config (already done) and wire it globally:**

**src/app.module.ts**

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

**import { ConfigModule, ConfigService } from '@nestjs/config';**

**@Module({**

**imports: [**

**ConfigModule.forRoot({**

**isGlobal: true, // makes ConfigService available everywhere**

**}),**

**// Example: TypeORM/Mongoose/etc. can read from ConfigService now**

**],**

**})**

**export class AppModule {}**

**Example usage in a service**

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

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

**@Injectable()**

**export class DbConfigService {**

**constructor(private cfg: ConfigService) {}**

**get user() { return this.cfg.get<string>('DB\_USER'); }**

**get pass() { return this.cfg.get<string>('DB\_PASS'); }**

**get host() { return this.cfg.get<string>('DB\_HOST'); }**

**get name() { return this.cfg.get<string>('DB\_NAME'); }**

**}**

**Example: TypeORM (if you use it)**

**// app.module.ts (excerpt)**

**import { TypeOrmModule } from '@nestjs/typeorm';**

**@Module({**

**imports: [**

**ConfigModule.forRoot({ isGlobal: true }),**

**TypeOrmModule.forRootAsync({**

**useFactory: () => ({**

**type: 'postgres',**

**host: process.env.DB\_HOST,**

**database: process.env.DB\_NAME,**

**username: process.env.DB\_USER,**

**password: process.env.DB\_PASS,**

**port: 5432,**

**autoLoadEntities: true,**

**synchronize: false,**

**}),**

**}),**

**],**

**})**

**export class AppModule {}**

**Because we applied Vault secrets before creating the app, process.env.DB\_\* values are already set when TypeORM initializes.**

**7) Run it**

**npm run start:dev**

**If you added a basic controller returning env, you can verify:**

* **GET http://localhost:3000/ → prints DB user, etc.**