Multi-tenant architecture for saas

1. **Định nghĩa**
    - Một ứng dụng/phần mềm phục vụ nhiều khách hàng (tenant) trên cùng một hệ thống.
    - Mỗi tenant có dữ liệu, người dùng, cấu hình riêng, nhưng dùng chung codebase và hạ tầng.

2. **Lợi ích**
    - **Tiết kiệm chi phí:** Chia sẻ tài nguyên, giảm chi phí vận hành và bảo trì.
    - **Dễ mở rộng:** Thêm tenant mới nhanh chóng, không cần triển khai lại ứng dụng.
    - **Quản lý tập trung:** Cập nhật, bảo trì, backup, restore chỉ cần thực hiện một lần cho toàn hệ thống.
    - **Tùy biến:** Cho phép mỗi tenant tùy chỉnh cấu hình, giao diện, người dùng riêng.

3. **Nhược điểm**
    - **Rủi ro bảo mật:** Dữ liệu các tenant nằm chung hệ thống, cần kiểm soát truy cập chặt chẽ.
    - **Ảnh hưởng lẫn nhau:** Tenant sử dụng nhiều tài nguyên có thể ảnh hưởng hiệu năng các tenant khác (“noisy neighbor”).
    - **Khó phân quyền sâu:** Quản lý quyền truy cập, phân tách dữ liệu phức tạp hơn single-tenant.

4. **Các mô hình triển khai Multi-Tenancy**
    - **Single Database, Shared Schema:**
        - Tất cả tenant dùng chung bảng, phân biệt bằng tenant_id.

    - Dễ triển khai, chi phí thấp, nhưng bảo mật và backup/restore phức tạp.
- **Single Database, Separate Schema:**
    - Mỗi tenant có schema riêng trong cùng database.
    - Dễ quản lý, bảo mật tốt hơn, backup/restore từng tenant dễ hơn.
- **Multiple Database:**
    - Mỗi tenant có database riêng.
    - Bảo mật cao nhất, dễ backup/restore, nhưng chi phí và quản lý phức tạp.

5. **Các mô hình SaaS Multi-Tenant phổ biến**
    - **URL-based:** Mỗi tenant có URL riêng.
    - **Shared App, Shared DB:** Dùng chung app, chung DB, phân biệt bằng tenant_id.
    - **Virtualization-based:** Mỗi tenant chạy trên VM/container riêng, tăng cách ly.

6. **Best Practices**
    - Thiết kế kiến trúc có khả năng mở rộng ngay từ đầu (cloud-native, auto-scaling).
    - Tự động hóa provisioning, deployment (CI/CD, IaC).
    - Giám sát, logging chi tiết cho từng tenant.
    - Cho phép tùy biến qua cấu hình, API, module mở rộng.
    - Lên kế hoạch backup, restore, và migration dữ liệu cho từng tenant.

7. **Tiêu chí chọn mô hình tenancy**
    - **Scalability:** Số lượng tenant, dung lượng lưu trữ mỗi tenant, tổng dung lượng, workload.
    - **Tenant isolation:** Độ tách biệt dữ liệu và hiệu năng giữa các tenant.

- **Per-tenant cost:** Chi phí database cho từng tenant.
- **Development complexity:** Độ phức tạp khi thay đổi schema, query.
- **Operational complexity:** Quản lý hiệu năng, backup, restore, disaster recovery, schema management.
- **Customizability:** Dễ dàng hỗ trợ schema riêng cho từng tenant hoặc nhóm tenant.

8. **Lưu ý khi chọn mô hình database**
    - **Single DB, Shared Schema:**
        - Đơn giản, dễ triển khai, phù hợp cho MVP hoặc số lượng tenant nhỏ/vừa.
        - Cần dùng global query filter (EF Core, Dapper) hoặc row-level security để tránh lộ dữ liệu giữa các tenant.
        - Luôn kiểm tra kỹ các truy vấn, nhất là khi dùng stored procedure, Dapper, hoặc query phức tạp (rất dễ bị oversharing data nếu quên filter).
        - Nên có integration test kiểm tra oversharing giữa các tenant.
    - **Database per tenant:**
        - Tối ưu bảo mật, dễ tách tenant lớn ra riêng, phù hợp cho khách hàng enterprise hoặc yêu cầu compliance cao.
        - Việc migrate schema/update cho hàng trăm/thousands DB sẽ rất phức tạp, cần automation tốt (Flyway, Liquibase, custom tool).
        - Có thể chia nhỏ: tenant nhỏ dùng chung DB, tenant lớn có DB riêng.
        - Nếu dùng Azure/AWS, có thể tận dụng elastic pool/sharding/cell để phân bổ tài nguyên.
    - **Hybrid/Cell-based:**
        - Chia tenant thành các nhóm (cell/shard), mỗi nhóm dùng 1 DB, giúp cân bằng giữa quản lý và hiệu năng.
        - Dễ scale, dễ di chuyển tenant giữa các cell khi cần.

9. **Kinh nghiệm vận hành**
    - **Đừng tối ưu quá sớm:** Bắt đầu với mô hình đơn giản nhất, chỉ chuyển sang mô hình phức tạp khi thực sự cần (scale, compliance, khách hàng lớn).
    - **Thiết kế modular:** Đảm bảo codebase dễ chuyển đổi giữa các mô hình (single DB, multi DB, cell).
    - **Luôn lưu tenantId trong JWT hoặc context:** Đảm bảo mọi truy vấn đều filter theo tenantId.
    - **Tách biệt config/metadata:** Nên có master DB lưu thông tin tenant, cấu hình, user, billing, v.v.
    - **Chuẩn bị cho việc migrate tenant:** Khi tenant lớn lên hoặc có nhu cầu riêng, cần dễ dàng chuyển sang DB riêng hoặc server riêng chỉ bằng thay đổi connection string.

10. **Vấn đề performance & scaling**
    - **Noisy neighbor:** Tenant lớn có thể ảnh hưởng tenant nhỏ nếu dùng chung DB/server. Nên có quota, throttle, hoặc tách DB khi cần.
    - **Index, partition:** Cần tối ưu index theo tenantId, có thể partition table nếu DB hỗ trợ.
    - **Caching, queue, storage:** Ngoài DB, cần nghĩ đến cách partition cache (Redis), queue (RabbitMQ), file storage cho từng tenant.

11. **EF Core Support cho các mô hình:**
    - **Discriminator (tenant_id column):**
        - Dùng global query filter để tự động filter dữ liệu theo tenant.
        - Hạn chế lỗi oversharing do quên filter trong code.

```
protected override void OnModelCreating(ModelBuilder modelBuild
er)
    ⇒ modelBuilder.Entity<MultitenantContact>()
        .HasQueryFilter(mt ⇒ mt.Tenant == _tenant);
```

- **Database per tenant:**
  - Chỉ cần truyền đúng connection string cho từng tenant.
  - Dễ dàng tách tenant lớn ra riêng, hoặc chuyển đổi khi cần.

```
{
 "Logging": {
  "LogLevel": {
   "Default": "Information",
   "Microsoft.AspNetCore": "Warning"
  }
 },
 "ConnectionStrings": {
  "TenantA": "Data Source=tenantacontacts.sqlite",
  "TenantB": "Data Source=tenantbcontacts.sqlite"
 },
 "AllowedHosts": "*"
}
```

- **Schema per tenant:**
  - EF Core không hỗ trợ tốt cho multi-schema, cần override schema trong OnModelCreating nếu muốn dùng.

12. **Đăng ký DbContextFactory đúng lifetime:**
    - **Single DB:**
      - Dùng Scoped lifetime cho DbContextFactory để mỗi user/session có config riêng.
    - **Multi DB:**
      - Nếu user có thể chuyển tenant trong session, dùng Transient lifetime để mỗi lần tạo DbContext sẽ lấy lại connection string mới.

13. **Tenant Service Pattern:**
    - Tạo ITenantService để lưu và quản lý tenant hiện tại.

```
namespace Common
{
   public interface ITenantService
   {
      string Tenant { get; }

      void SetTenant(string tenant);

      string[] GetTenants();

      event TenantChangedEventHandler OnTenantChanged;
   }
}
```

- Inject ITenantService vào DbContext để filter hoặc chọn connection string phù hợp.

```
public ContactContext(
   DbContextOptions<ContactContext> opts,
   ITenantService service)
   : base(opts) ⇒ _tenant = service.Tenant;
```

Single database

```
public ContactContext(
   DbContextOptions<ContactContext> opts,
   IConfiguration config,
   ITenantService service)
   : base(opts)
{
   _tenantService = service;
   _configuration = config;
}
```

Multiple databases

14. **Migration & Schema Update:**
    - Với multi-db, việc migrate schema/update cho nhiều DB cần automation tốt.
    - Nên dùng tool như Flyway, Liquibase, hoặc custom migration runner.

15. **Elastic Pool & Automation**
    - **Elastic pool:** Cho phép chia sẻ tài nguyên giữa nhiều DB, giảm chi phí mà vẫn đảm bảo hiệu năng từng tenant.
    - **Automation:** Quản lý hàng ngàn DB (backup, restore, index, HA, encryption, telemetry) cần script/devops tự động hóa.

Summary: SaaS là gì?

- **SaaS (Software as a Service)** là mô hình cung cấp phần mềm qua internet, nơi khách hàng (tenant) sử dụng ứng dụng mà không cần cài đặt hay quản lý hạ tầng.
- Khách hàng trả phí theo thời gian sử dụng (thường là thuê bao tháng/năm).
- Ứng dụng SaaS thường chạy trên nền tảng cloud, hỗ trợ nhiều tenant cùng lúc (multi-tenancy).
- Mỗi tenant có dữ liệu, người dùng, cấu hình riêng, nhưng dùng chung codebase và tài nguyên hệ thống.
- Ưu điểm: dễ tiếp cận, cập nhật nhanh, tiết kiệm chi phí, mở rộng linh hoạt, quản lý tập trung.
- Ví dụ: Office 365, Salesforce, Google Workspace, Slack, Zoom.

References

- **Wikipedia**: Multitenancy
- **Viblo**: Multi-tenacy và kiến trúc database cho hệ thống multi-tenacy
- **frontegg**: Multi-Tenant Architecture: How It Works, Pros, and Cons
- **frontegg**: Multi-Tenant SaaS: Benefits and Best Practices

- **Learn.Microsoft**: Multi-tenancy
- **Learn.Microsoft**: Multitenant SaaS database tenancy patterns
- **Learn.Microsoft**: SaaS and multitenant solution architecture
- **Reddit**: Good multi-tenant architecture for saas
- **DeveloperPartners**: Multi-Tenant SaaS Architecture - Database Per Tenant