Agentic SaaS Platform with Multi-Tenancy

Standard SaaS Multi-Tenancy Approach

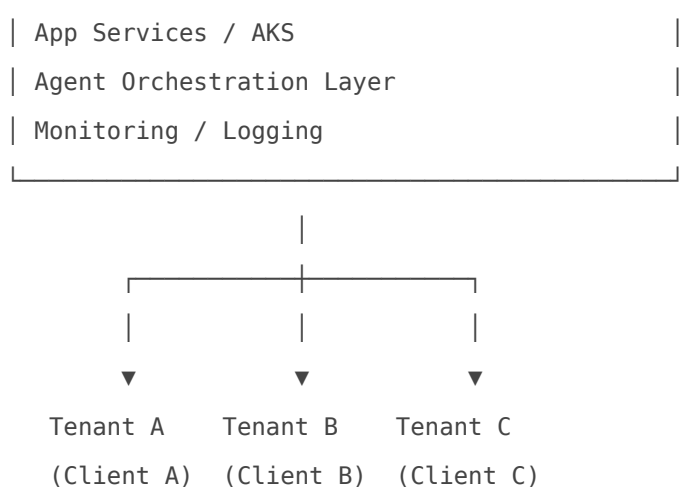
> Note: This is generated content. Please refer valid sources while implementing.

What is Multi-Tenancy?

Multi-tenancy is a software architecture pattern in which a single application platform serves multiple customers (tenants) while ensuring that each tenant's users, data, configurations, sessions, and workloads remain isolated from those of other tenants. A tenant typically represents a customer organization, business unit, or client. Although tenants may share parts of the underlying infrastructure (such as application services, compute clusters, or monitoring systems), the platform enforces strict logical or physical boundaries to prevent unauthorized access, data leakage, and performance interference across tenants.

A common enterprise SaaS architecture uses **shared infrastructure with logical tenant isolation**, while introducing stronger isolation controls when required for compliance, regulatory, security, or performance reasons.

High-Level Architecture



The application serves multiple customers (tenants) from the same platform while ensuring that data, sessions, and workloads remain isolated.

---

1. Identity Isolation

Users authenticate through an Identity Provider such as **Microsoft Entra ID**.

```
User Login
   ↓
Entra ID
   ↓
JWT Token
   ↓
Tenant Claim
   ↓
Application Authorization
```

The platform determines:

- Which tenant the user belongs to
- What resources the user can access
- Which data the user can view
- Which agents or services the user can invoke

Note: Entra Tenant ID may be used for authentication and tenant identification, but it is typically only one part of the overall tenant isolation strategy.

---

2. Data Isolation

Model A: Shared Database (Most Common)

All tenants share the same database, with records tagged using a tenant identifier.

```
Users

TenantId | UserId | Name
A        | 101    | John
B        | 102    | Mike
```

All queries are filtered by tenant context.

```
SELECT *
FROM Documents
WHERE TenantId = @CurrentTenant
```

Pros

- Lower cost
- Easier operations
- Better resource utilization

Cons

- Requires strong authorization and access controls

Model B: Database Per Tenant

```
Client A → DB_A
Client B → DB_B
Client C → DB_C
```

Pros

- Strong isolation

- Easier compliance discussions

Cons

- Higher operational overhead
- Increased cost

Model C: Hybrid Model

Very common in enterprise SaaS platforms.

```
Small / Standard Clients
        ↓
Shared Database

Regulated / Premium Clients
        ↓
Dedicated Database
Dedicated Storage
```

---

3. Agent Session Isolation

For AI-enabled platforms, agent isolation is critical.

Each session is scoped using:

```
TenantId
UserId
SessionId
```

This ensures:

- Conversation history remains isolated
- Agent memory is tenant-specific
- Runtime context cannot leak across clients
- Prompts and responses are tenant-bound

Example:

```
Client A
 └─ User A
     └─ Session A1

Client B
 └─ User B
     └─ Session B1
```

Agent memory from Client A should never be accessible to Client B.

---

4. Vector Store / RAG Isolation

In GenAI architectures, embeddings and indexed documents must be isolated.

Shared Vector Store with Partitions

```
Vector Index
├── Tenant A
├── Tenant B
└── Tenant C
```

Dedicated Vector Stores

```
Vector DB A
Vector DB B
Vector DB C
```

This prevents retrieval of another tenant's documents during RAG (Retrieval-Augmented Generation).

---

5. Compute Isolation

Application compute is often shared.

```
AKS Cluster
 ├─ Tenant A Requests
 ├─ Tenant B Requests
 └─ Tenant C Requests
```

Isolation is typically achieved using:

- Kubernetes namespaces
- Resource quotas
- Rate limiting
- Autoscaling
- Workload scheduling policies

For high-security or premium customers:

```
Dedicated AKS Cluster
or
Dedicated Deployment
```

may be provided.

---

6. Storage Isolation

Shared Storage

```
Storage Account
 ├─ tenant-a/
 ├─ tenant-b/
 └─ tenant-c/
```

Dedicated Storage

```
Storage Account A
Storage Account B
Storage Account C
```

The chosen model depends on security, compliance, and customer requirements.

7. Shared vs Dedicated Components

Shared Components

- API Gateway
- Application Services
- AKS/App Services
- Monitoring & Logging
- CI/CD Pipelines
- Agent Orchestration Services
- Identity Integration Components

Potentially Dedicated Components

- Databases
- Blob Storage
- Key Vaults
- Vector Databases
- AI Models/Deployments
- Compute Resources

Typical Enterprise Architecture Response

> In a standard SaaS multi-tenant architecture, a client (tenant) is isolated through application-level tenant controls rather than completely separate infrastructure. User identity is mapped to a tenant, and all data access, agent sessions, vector indexes, and workload execution are scoped to that tenant. Infrastructure such as API gateways, application services, Kubernetes clusters, and monitoring platforms is often shared across tenants, while customer data stores, storage accounts, vector databases, or compute resources may be logically or physically

> separated depending on security, compliance, and performance requirements. Entra Tenant ID may be used for identity federation, but the primary concern is application-level tenant isolation across data, sessions, and workloads.

Revision #1
Created 2026-07-10 14:35:56 UTC by Soubhik Mazumdar
Updated 2026-07-10 14:39:57 UTC by Soubhik Mazumdar