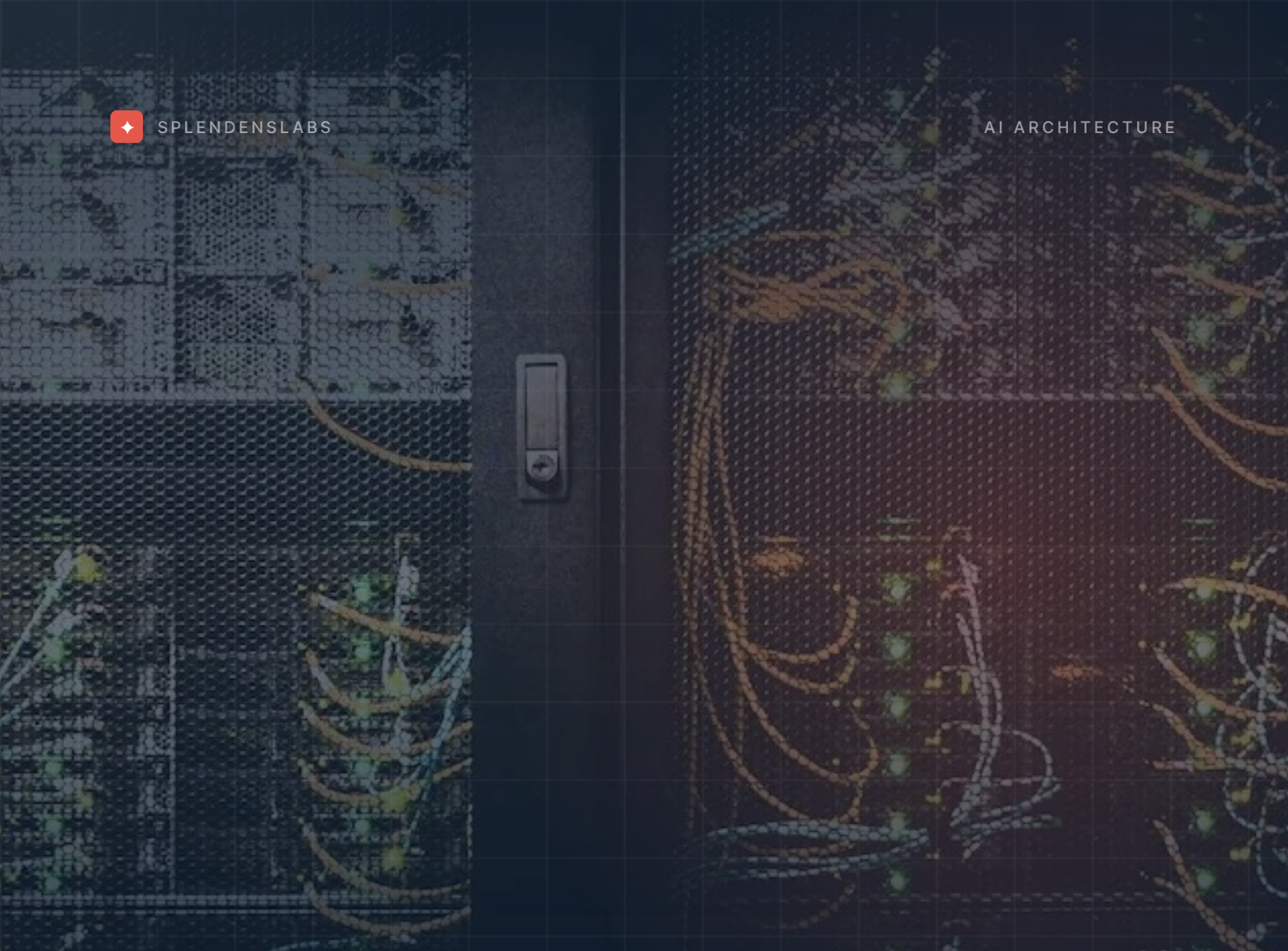


✦ REFERENCE ARCHITECTURE · AI ARCHITECTURE

Multi-Tenant AI SaaS Architecture

Isolation models, per-tenant indexes and keys, shared inference, quotas, cost attribution and data residency for AI products that serve many customers from one platform.

FORMAT	LEVEL	READ TIME	UPDATED
Reference Architecture	Advanced	20 min	Mar 2026

OVERVIEW

What this reference architecture covers

Adding AI to a multi-tenant product introduces new places for tenants to leak into each other: shared vector indexes, shared caches, shared prompts and a shared, expensive inference budget. This reference architecture shows how to choose an isolation model per resource, carry tenant context through every layer, and keep one tenant's data, load and bill from affecting another.

KEY TAKEAWAYS

- Isolation is a per-resource decision: the same product can pool compute, bridge indexes and silo encryption keys.
- Tenant context must be resolved once at the edge and enforced in every query, cache key, index filter and trace.
- Vector indexes and caches are the most common cross-tenant leak paths — partition or filter them by tenant by construction.
- Shared inference needs per-tenant quotas, fair scheduling and priority tiers, or one large customer degrades everyone.
- Attribute every token, embedding and GPU-second to a tenant from day one; retrofitting cost attribution is painful.
- Data residency and per-tenant keys are easier to design in than to migrate to later.

WHO IT'S FOR

SaaS CTOs and platform architects

AI and backend engineers

Security and compliance leads

Product and finance owners setting pricing

HOW TO USE IT

Use section 01 to choose an isolation model per resource, 02–04 to design the data and inference paths, and the checklist in 06 as a review gate before onboarding enterprise tenants.

CONTENTS

01 Choosing an isolation model
02 The reference architecture
03 Per-tenant indexes, caches and keys
04 Shared inference, quotas and noisy neighbors
05 Cost attribution and tenant configuration
06 Data residency and the readiness checklist

01 / ISOLATION

Choosing an isolation model

There is no single right model. Choose per resource, based on the sensitivity of the data, the noisiness of the workload and what your largest customers will contractually require.

MODEL	WHAT IS SHARED	STRENGTHS	COSTS
Pooled	Everything; rows tagged with tenant ID	Cheapest, simplest to operate, fast onboarding	Isolation depends entirely on correct filtering; noisy neighbors
Bridge	Compute shared; data partitioned per tenant (schema, index, namespace)	Stronger data boundary, per-tenant backup and delete	More objects to manage; migrations run per partition
Silo	Nothing; dedicated stack or account per tenant	Strongest isolation, custom residency and keys	Highest cost; deploys and upgrades multiply

COMMON DEFAULT

Pooled compute

Stateless API and orchestration services are shared and scale horizontally; tenant context travels with each request.

COMMON DEFAULT

Bridged data

Each tenant gets its own index namespace, schema or prefix so a missing filter cannot return another tenant's rows.

ENTERPRISE TIER

Siloed on demand

Dedicated deployment, region or keys offered as a premium tier for tenants whose contracts require it.

DESIGN NOTE

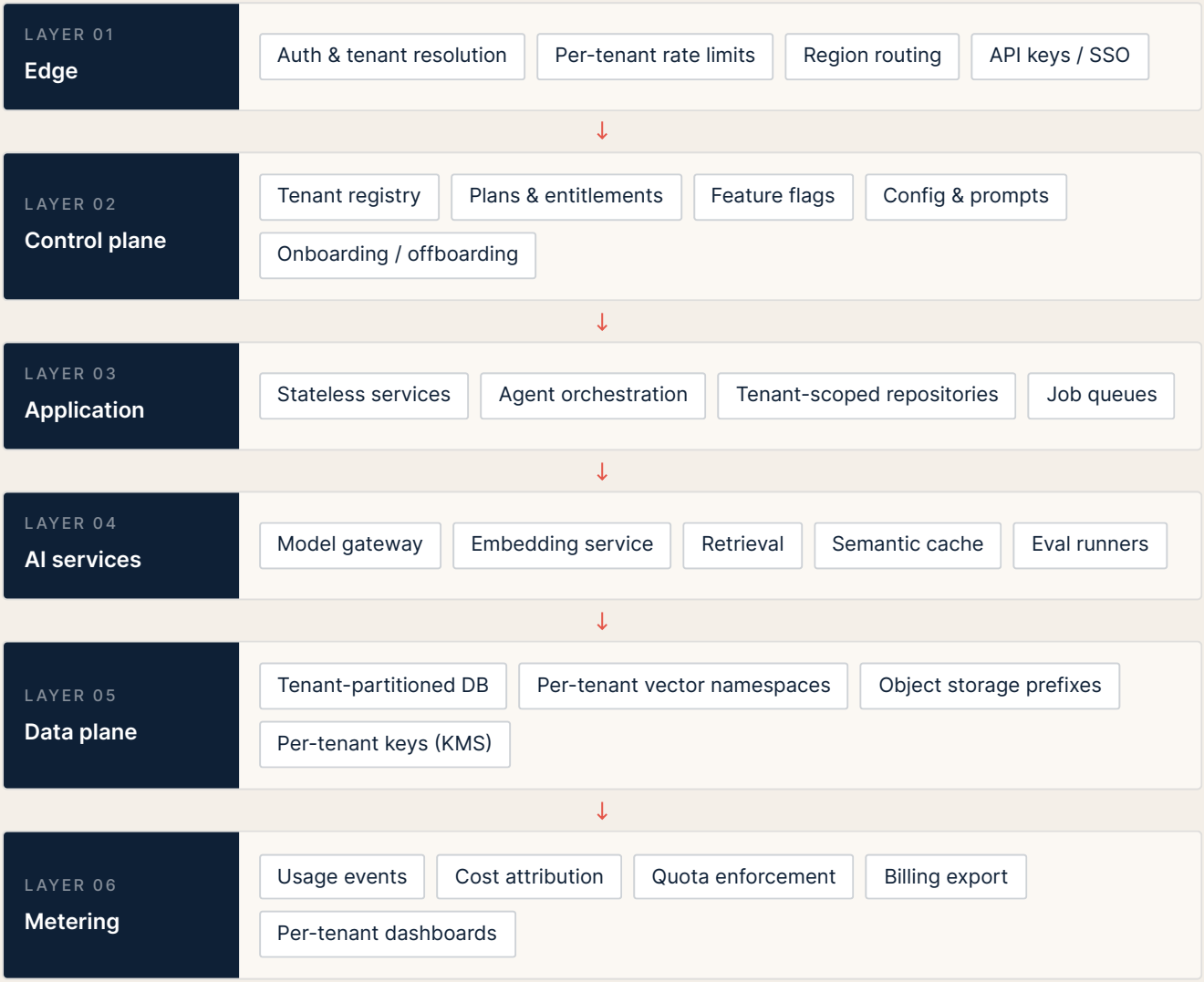
Build one codebase for all tiers

Pooled, bridged and siloed tenants should run the same code with different configuration. A separate fork for the enterprise silo becomes a second product to maintain.

✦ 02 / ARCHITECTURE

The reference architecture

Tenant context is resolved once at the edge and then carried, never re-derived, through every layer below it.



Carrying tenant context

01 **Resolve at the edge**

Derive the tenant from the authenticated identity or API key — never from a request parameter the client controls.

02 **Propagate in context**

Pass tenant ID in request context and trace baggage so every downstream call, job and span carries it.

03 **Enforce in the data layer**

Use row-level security, tenant-scoped repositories or per-tenant connections so queries cannot omit the filter.

04 **Include in every key**

Cache keys, index namespaces, object paths and queue partitions all start with the tenant ID.

> If the tenant ID is optional anywhere, it will be missing somewhere.
>
> SPLENDENSLABS ENGINEERING PRINCIPLE

✦ 03 / DATA

Per-tenant indexes, caches and keys

AI features add stores that traditional SaaS did not have. Each one needs a tenant boundary as deliberate as the main database's.

STORE	LEAK RISK	RECOMMENDED BOUNDARY
Vector index	Similarity search returns nearest chunks from any tenant	Namespace or collection per tenant; tenant filter enforced server-side
Response / semantic cache	Cached answer served to a different tenant	Tenant ID (and permission scope) as the first component of the key
Prompt & few-shot store	Tenant examples bleed into other tenants' prompts	Tenant-scoped examples; shared ones curated and free of customer data
Fine-tuned models / adapters	Model memorizes one tenant's data	Per-tenant adapters only with consent; never train shared models on tenant data by default
Traces & logs	Support staff see prompts across tenants	Tenant tag on every span; access controls and redaction in the trace store

01

Envelope encryption per tenant

Encrypt tenant data with a data key wrapped by a per-tenant key in a KMS. Revoking the tenant key renders its data unreadable.

02

Bring your own key

Enterprise tenants can supply the wrapping key from their own KMS; design the key lookup so this is configuration, not code.

03

Clean offboarding

Deleting a tenant removes its namespace, cache entries, object prefix, adapters and keys — tested as a scripted job.

04

Embeddings are data

Treat vectors as derived customer data: same residency, retention and deletion rules as the source text.

✦ 04 / INFERENCE

Shared inference, quotas and noisy neighbors

Model capacity — provider rate limits or your own GPUs — is the scarcest shared resource. Without controls, one tenant's batch job becomes everyone's outage.



SINGLE SHARED QUEUE	TIERED, FAIR ADMISSION
First come, first served	Per-tenant token and request quotas
A bulk import starves interactive users	Separate interactive and batch lanes
Provider rate-limit errors hit random tenants	Weighted fair scheduling across tenants
No way to honor premium SLAs	Premium tiers get reserved capacity

Quota controls

- [ ] Requests per minute and tokens per minute per tenant, per model
- [ ] Monthly spend or usage cap per tenant, with soft-limit warnings before hard stops
- [ ] Concurrency limit per tenant for long-running agent runs
- [ ] Batch and background work routed to a lower-priority lane
- [ ] Clear, documented error and retry-after behavior when a tenant hits its limit

05 / ECONOMICS

Cost attribution and tenant configuration

AI makes gross margin vary per tenant. You need to know what each tenant costs before you can price, package or throttle sensibly.

What to meter per tenant

RESOURCE	UNIT	CAPTURED AT
LLM calls	Input, cached and output tokens by model	Model gateway
Embeddings	Tokens embedded; re-index jobs	Embedding service
Vector storage	Vectors stored and dimensions	Nightly index snapshot
Self-hosted GPU	GPU-seconds, apportioned by tokens served	Inference server metrics
Tools & third-party APIs	Calls and vendor cost	Tool adapters

CONFIG

Layered settings

Platform defaults, plan-level overrides and tenant overrides, resolved in that order and cached per request.

CONFIG

Per-tenant prompts

Tenants can customize tone, glossary and instructions within guardrails; versioned and evaluated like code.

CONFIG

Model allowlists

Tenants may restrict which providers or regions process their data; the gateway enforces it per request.

06 / COMPLIANCE

Data residency and the readiness checklist

Residency covers more than the primary database. Prompts, embeddings, caches, traces and provider calls all move data.

Residency design rules

- Pin each tenant to a home region in the tenant registry and route at the edge.
- Deploy the data plane, vector store, cache and trace store per region; share only the control plane metadata that contains no customer content.
- Use model endpoints in the tenant's region, or document which providers process data where; keep backups and eval datasets in-region too.

ISOLATION

- [ ] Isolation model chosen and documented per resource
- [ ] Automated test proves tenant A cannot read tenant B through API, search or cache
- [ ] Tenant ID never accepted from client input

DATA

- [ ] Per-tenant vector namespaces and cache keys
- [ ] Per-tenant encryption keys; BYOK path available
- [ ] Tenant deletion job removes every store, including embeddings and traces

INFERENCE

- [ ] Per-tenant quotas and spend caps enforced at the gateway
- [ ] Interactive and batch lanes separated
- [ ] Usage metered per tenant, per model

OPERATIONS

- [ ] Every span tagged with tenant and region; per-tenant cost dashboards
- [ ] Residency verified for prompts, embeddings, caches and logs

✦ GET IN TOUCH

Let's Talk

We build multi-tenant AI platforms and run our own across therapy, education, pet care, commerce and CRM. Bring us your tenancy model and your largest customer's security questionnaire.

- Isolation model review per resource
- Tenant-aware RAG, caching and inference
- Metering, quotas and cost attribution

BOOK AN AI STRATEGY SESSION →

EMAIL
hello@splendenslabs.com

WEB
splendenslabs.com

OFFICE
Sona Incubation Foundation, Salem