— CHAPTER 11 - MULTI-TENANT HAPI - COLD OPEN

Last chapter we sketched the interceptor. Now we wire it for production.

One HAPI server, many clinics. Multi-tenancy is the feature that turns a single FHIR server into a platform - and the one place where a single bug silently shows Clinic A another clinic's patients. So we treat isolation as the thing we must never get wrong.

Three goals for this chapter

- Choose a HAPI partitioning mode that isolates clinics
- Wire a tenant registry - clinic slug to partition id
- Recognise two cross-tenant footguns before they leak

— CONCEPT - THE PRODUCTION REALITY

One server. Dozens of clinics. One requirement that can never break.

We don't run one HAPI server per clinic. In Dudoxx HMS we run a single server, and every clinic is a tenant living inside it. That buys operational simplicity - and takes on the hardest correctness requirement in the system.

FORCE - 1

One deployment

A single server, one ops surface, one backup, one upgrade. The whole reason to do this instead of N servers.

FORCE - 2

N tenants

Each clinic is a tenant, onboarded without standing up new infrastructure.

FORCE - 3

Isolation

The non-negotiable. Clinic A's queries must never return Clinic B's patients - under any circumstance.

- **HAPI's native mechanism for this is partitioning**
- **Isolation is the one requirement we must never get wrong**

— COMPARISON - HOW HAPI ASSIGNS A REQUEST TO A PARTITION

Two shapes. They differ only in where the tenant identity is read.

HAPI assigns every request to a partition, and the assignment is interceptor-driven - the same mechanism from Chapter 10. The interceptor is the seam; the mode just changes what it reads.

— BUILT-IN - PATH-TENANT

POST /fhir/P1/Patient

RequestTenantPartitionInterceptor paired with UrlBaseTenantIdentificationStrategy. The tenant becomes a URL path segment - P1 is the tenant. Clean, built-in, no custom code.

— OUR PATH - HEADER-TENANT

X-Clinic-ID: northgate

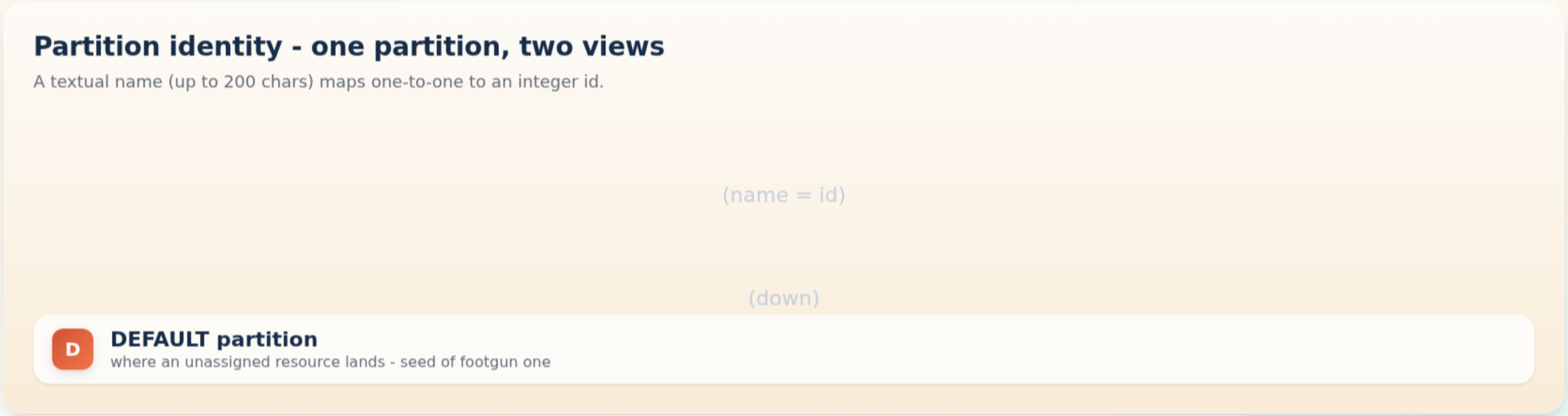
Keep the partition interceptor, but read the tenant from a request header instead of the path. Our clinic identity already lives in a header the gateway sets - and we keep tenants out of every FHIR URL.

- **We take the header path - gateway already sets X-Clinic-ID**
- **Either way, the interceptor is the seam**

— DIAGRAM - PARTITIONSETTINGS - NAME TO ID

Partitioning is off by default. Once on, every row gets a partition id.

You enable partition-aware storage through PartitionSettings, before the server initialises. Once on, HAPI adds a PARTITION_ID integer column to every resource table and stamps each row with the partition it belongs to.



Verify - exact enable keys live in the Chapter 7 config dossier - pin your tag, we don't quote yaml here

— COMPARISON - THE READ POINTCUT DECIDES EVERY TIME

The whole isolation guarantee rests on one return value.

Reads go through STORAGE_PARTITION_IDENTIFY_READ - it returns the partition or partitions a search is allowed to touch. Return one, return many: same hook, opposite outcomes.

— RETURNS ONE NAME - PARTITION-SCOPED

The clinic sees only its own data

The caller's partition, and nothing else. This is the true tenant isolation we promised - what a clinic user must always get.

— RETURNS MANY - CROSS-PARTITION

HAPI searches across all of them

A catastrophic leak for a clinic user - but the legitimate feature for an internal admin or a reporting job that spans tenants. Both, depending on who's asking.

- **Isolation = READ hook returns exactly the caller's partition, nothing more**
- **Verify - cross-partition reference behaviour is version-sensitive**

— WALKTHROUGH - CALLBACK TO CHAPTER 10

Remember the ClinicPartitionInterceptor? This is where it earns its keep.

It overrides the two Pointcuts we just described. Both hooks do the same small job - produce a partition id - then hand it back to HAPI, which does the actual stamping or scoping. The interceptor itself stays thin.

— HOOK - STORAGE_PARTITION_IDENTIFY_CREATE

Fires on every write

Answers one question - which partition does this new resource belong to? Reads the tenant identity off the request, returns a partition id.

— HOOK - STORAGE_PARTITION_IDENTIFY_READ

Fires on every read and search

Answers - which partition or partitions may this query see? Same shape, returns a partition id to scope the search.

- **The hook does NOT know how to turn a slug into an id - it delegates**
- **Two hooks, one job each, both return a partition id**

— CONCEPT - THE PATTERN WE BUILD - NOT A HAPI CLASS

A slug to id lookup on every request needs to be fast. So we cache.

The interceptor needs to turn a clinic slug into a partition id on every call. A database hit per request is too slow - so we put a registry in front of it. This is our pattern; HAPI gives the hooks, the registry is the glue we write.

TenantRegistryService - slug to id with a ~60s TTL

Almost every request is a cache hit with no database cost.

(down)

(down)

3 **partitionId returned**
60s TTL - a newly onboarded clinic is visible within a minute

- **The registry is the bridge from a human slug to HAPI's integer id**
- **Our pattern to build - HAPI gives the hooks**

— DIAGRAM - ONE WRITE, ALL THE WAY THROUGH

Header in. Partition id out. The Patient belongs to Northgate, at the storage layer.

A client posts a new Patient to /fhir/Patient with header X-Clinic-ID set to "northgate". Follow it through the machine.

POST /fhir/Patient - X-Clinic-ID: northgate

CREATE hook to registry to Pointcut to stamped row.

(next)

(next)

3 **HAPI stamps PARTITION_ID = 7**
the Patient now belongs to Northgate, permanently

- **Header missing or slug unknown - REJECT the request, return an error**
- **Never fall back to DEFAULT - that silently drops it in a shared bucket**

— COMPARISON - TWO LEVELS OF ISOLATION - REAL SYSTEMS MIX THEM

Isolate physically where you must. Logically everywhere else.

Partitions inside one database are not the only way to isolate tenants. There are two levels, with very different operational bills.

— LEVEL 1 - SHARED-SCHEMA PARTITIONING

One schema, isolated by PARTITION_ID

Operational simplicity - one connection pool, one set of migrations, one backup. The cost is shared blast radius: a noisy tenant affects everyone, and a buggy READ hook can leak across tenants.

— LEVEL 2 - PER-TENANT DATABASE

Physical separation per clinic

Hard separation for data-residency mandates, independent scaling and backup. You pay for it - N migrations, N pools, and no native cross-tenant search, because the data genuinely lives apart.

- **Production middle ground - partitions for routine tenancy**
- **Dedicated DB only for tenants with an explicit residency mandate**

— CONCEPT - TWO FOOTGUNS - BOTH SILENT

No error. No warning. Just leaked data, discovered weeks later.

Both are dangerous precisely because nothing throws. You find them by auditing - or by a customer noticing someone else's patient.

— FOOTGUN 1 - DEFAULT-PARTITION LEAK

Forget to assign - it lands in DEFAULT

Forget a partition, or let a registry miss fall through instead of rejecting, and the resource lands in DEFAULT. Anything in DEFAULT is visible to any query that isn't strictly partition-scoped. You've effectively published it. Reject, never default.

— FOOTGUN 2 - CROSS-TENANT QUERY BUG

READ hook returns more than the caller's partition

Return a list, return null, return "all" by mistake - and every clinic can suddenly see every other clinic. One wrong return value in one hook, and the entire isolation guarantee is gone.

- **Verify - confirm HAPI 8.x default and null-partition behaviour against your pinned tag**
- **Write a test proving a scoped read can't see the default bucket**

— DIAGRAM - THE WHOLE MACHINE IN ONE PATH

One header in. Correct isolation out. Every box is a slide we just walked.

Five numbered steps tie the chapter together. If you can draw this from memory, you can wire multi-tenant HAPI.

Request to partition - end to end

Cross-references slides 03, 06, 07, 08, 04.

(down)

(down)

3 **Registry resolves slug to id**
from its 60s cache - slide 07

(down)

4 **id flows back to the Pointcut**
the partition id HAPI works in - slide 08

(down)

5 **HAPI JPA does the work**
stamps PARTITION_ID on write - scopes the search on read - slide 04

— TRY IT YOURSELF - PROVE YOUR OWN ISOLATION

Register two clinics. Post a Patient to each. Then drop the header.

Enable partitioning in your pinned HAPI, register northgate and southbay in your tenant table, then run the test that matters.

— THE EXERCISE

POST a Patient with each X-Clinic-ID, then a scoped GET

Confirm a scoped GET for one clinic never returns the other's Patient. Then the negative test - drop the header entirely and confirm the request is rejected, not parked in DEFAULT. If it lands in DEFAULT, you've reproduced footgun one.

Recap

- Header-tenant mode + the partition interceptor
- Tenant registry - slug to id, 60s cache
- Reject, never DEFAULT - scope every READ