DataSitr Architecture Overview

Generated: 2026-04-21T05:00:43Z

Source document: docs/architecture-overview.md

Git commit: c656136

Generator: [operator-tooling]

Benchmark artifact: docs/generated/pii_benchmark_latest.json (generated 2026-04-20T03:31:24Z, Arabic NER loaded, 1K p95 31.48 ms)

What DataSitr does

DataSitr is a privacy gateway that sits between applications and AI providers. It detects personal data in requests, applies privacy controls based on a three-lane policy engine, routes to the appropriate AI provider, and restores original data in responses. All processing decisions are recorded for PDPL compliance.

Request flow

[See source document for diagram/code]

Pipeline steps in order

Step	Module	What happens
1	`api/routes/process.py`	Authenticate request, extract tenant ID, assign request ID
2	`detector.py`	Scan text for PII using Presidio + Saudi pattern recognizers
3	`sensitive.py`	Classify any PDPL Article 1(11) sensitive categories (health, biometric, etc.)
4	`tokenizer.py`	Replace each PII entity with a typed placeholder (`[[PERSON:01]]`), store original in vault
5	`policy.py`	`decide_route()` evaluates detection confidence, sensitivity, tenant policy, use case
6	`providers.py`	`call_ai()` sends to the selected provider with retry + circuit breaker

7	`rehydrator.py`	Find placeholders in AI response, look up originals in vault, restore
8	`pipeline.py`	Rescan output for leaked PII, write compliance records, return result

---

Three-lane routing model

The policy engine (`policy.py`) classifies every request into one of four outcomes:

Green lane — external AI, detector-sanitized text

Conditions (all must be true):

- All detected PII entities have confidence >= threshold (default 0.85)
- No PDPL Article 1(11) sensitive data present
- Use case is in the safe-for-external list (summarize, classify, sentiment, etc.)
- Tenant policy allows external AI (`external_ai_enabled=True`)
- No high quasi-identifier risk remains after policy evaluation
- `check_true_anonymization()` confirms no original PII substrings remain in tokenized text

What happens:

- Detected PII is replaced with typed placeholders before eligible external processing
- Detector-sanitized placeholder text is sent to external AI (OpenAI, Anthropic, Google)
- Original detected PII values stay local in the vault
- Residual contextual re-identification risk is handled conservatively and still requires customer and legal review
- Transfer register entry written

Amber lane — in-Kingdom AI, pseudonymized

Conditions (any triggers amber):

- Mixed confidence scores (some entities below threshold)
- Legal exception path enabled by tenant's counsel
- Use case not on the safe-for-external list
- Tenant policy restricts to local but allows pseudonymized processing

What happens:

- PII replaced with pseudonymized placeholders
- Sent to in-Kingdom provider (STC SambaNova)
- PII mapping stays in vault, never crosses border

- No transfer register entry needed

Red lane — in-Kingdom AI, raw PII

Conditions (any triggers red):

- PDPL Article 1(11) sensitive data detected (health, biometric, criminal, etc.)
- Anonymization not possible or confidence too low
- Use case requires the AI to see real identifiers
- High-risk file type (OCR, voice transcript, etc.)

What happens:

- Raw text sent to in-Kingdom provider with full PII intact
- PII never leaves Saudi Arabia
- Processing record notes "local_raw" route

Block — request rejected

Conditions:

- `customer_forbids_external=True` and no in-Kingdom provider available
- Policy explicitly blocks the request
- Invalid or missing configuration

Tenant isolation

Every tenant's data is isolated at every layer:

Layer	Isolation mechanism
Vault	Per-tenant derived encryption key via HKDF-SHA256 by default; legacy rows may still use the older SHA-256 derivation. Cross-tenant retrieve returns `None`.
Auth	Each API key bound to a `customer_id`. All downstream operations scoped to that ID.
Logs	Processing records, transfer register, and billing all tagged with `tenant_id`.
Policy	`TenantPolicy` supports per-tenant overrides (external AI, legal exception, retention).
Rate limits	Independent rate limit buckets keyed by `api:{customer_id}`.

Cross-tenant access attempts are logged as `vault_unauthorized_access`.

See tenant-isolation.md for the full isolation model and invariants.

Request ID traceability

Every request gets a unique ID (client-supplied via `X-Request-ID` header, or auto-generated UUID). This ID propagates through:

1. HTTP response header (`X-Request-ID`)
2. Application logs
3. Processing record
4. Transfer register entry (if applicable)
5. Vault access log
6. Billing record

This allows tracing any processing event end-to-end from API request to compliance record.

Subject rights (PDPL)

DataSitr implements the following data subject rights:

Right	Implementation
Access	`export_subject_data(tenant_id, subject_identifier)` — returns all processing records and vault entries related to a subject
Erasure	`delete_subject_data(tenant_id, subject_identifier)` — deletes vault entries and logs the deletion event
Audit	All deletions logged to `deletion_log.jsonl` with timestamp, tenant, and subject identifier

Subject rights operations are scoped to the requesting tenant — a tenant cannot access or delete another tenant's subject data.

Provider architecture

[See source document for diagram/code]

Provider failures within a lane trigger the fallback chain. Cross-lane fallback never happens — a green-lane request will not silently fall back to red lane. If all providers in the assigned lane fail, the request returns an error (fail-closed).

Compliance record generation

Every processed request produces:

Record type	File	When
Processing record	`processing_records.jsonl`	Every request
Transfer register	`transfer_register.jsonl`	Green lane (external) requests only
Evidence pack	Inline in processing record	Every request

Records include: request ID, tenant ID, route decision, policy version, PII types detected, detection confidence, security measures applied, timestamp.

Retention: 5 years per SDAIA guideline. Records are append-only JSONL with auto-rotation at 5MB.

Data storage

Store	Engine	Encryption	Location
Token vault	SQLite in single-node mode; PostgreSQL in shared-state mode	AES-256-GCM per-tenant keys	`[internal file]` or PostgreSQL (`vault`)
Compliance logs	JSONL in file mode; PostgreSQL in shared-state mode	Raw PII excluded; sequenced rows can be signed for export	`data/compliance/` or PostgreSQL
API keys	JSON file in single-node mode; PostgreSQL in shared-state mode	SHA-256/HMAC hashed	`[internal file]` or PostgreSQL (`api_keys`)
Usage/billing	JSONL in single-node mode; PostgreSQL in shared-state mode	Not encrypted	`[internal file]` or PostgreSQL (`billing_usage_events`)
Tenant policies	JSON file in single-node mode; PostgreSQL in shared-state mode	Not encrypted	`[internal file]` or PostgreSQL (`tenant_policies`)
Webhook delivery log	JSONL in single-node mode; PostgreSQL in shared-state mode	Not encrypted	`[internal file]` or PostgreSQL (`webhook_delivery_log`)

Policy snapshots	JSON files in file mode; PostgreSQL in shared-state mode	Not encrypted	`data/compliance/policy_snapshots/` or PostgreSQL

The vault is the only store containing actual PII. It uses per-tenant encryption keys derived from the master key, with TTL-based auto-expiry (default 24 hours).

Module dependency graph

[See source document for diagram/code]

Related docs

- Threat Model — threat catalog and mitigations
- Tenant Isolation — isolation invariants and testing
- Security & Compliance — what DataSitr protects and how
- Versioning — API, policy, and schema versioning strategy

This document describes technical design intent and current operational posture. It does not constitute a warranty, service-level agreement, legal guarantee, or certification of regulatory compliance. DataSitr is designed to support PDPL alignment; it does not itself grant compliance. For the canonical list of safe and unsafe claims, contact gov@datasitr.com.