missing configuration artifacts to complete parameter coverage:

* Parameter Reference with a new section for **Insights**, including all relevant parameters drawn from docs 5.1–5.4.
* ETL execution parameters in a consistent format (as a section 6.4 or etl\_config.yml, aligned with the rest of the v14 documentation).
* all Insights‑related variables for the Cookiecutter stack (.env-compatible), ensuring completeness for CI/CD and infrastructure automation.

# **Insights Module – Parameter Reference (Konnaxion v14)**

## **Configuration Parameters and Invariants**

* **Analytical data retention:** All fact tables in the Insights (Reporting & Analytics) module use time-partitioning with fixed retention periods. For example, **Smart Vote** vote history is kept for **5 years**, **Usage** (MAU/projects/docs) for **10 years**, and **API performance** metrics for **2 years** – older partitions are dropped on a rolling basis. Materialized summary views always query only recent data (e.g. last 24 h or 30 d) and are refreshed rather than retaining historical states.
* **Backup policy:** The analytics PostgreSQL database ( “Reports” DB) is backed up with **nightly full dumps** plus **30‑minute incremental snapshots**, retained for **14 days**. (The Airflow metadata DB is included in these backups.) DAG definitions (code) are versioned in Git, and the **Redis** cache is **not** backed up (ephemeral usage). Quarterly restore drills are performed to ensure disaster-recovery readiness.
* **Cache TTL:** The Insights module leverages the same Redis cache service as the core platform for temporary analytics data. Cached query results (e.g. for dashboards) expire after **12 hours**, and a daily cleanup job purges any stale keys older than this TTL. This ensures the analytics caches stay fresh without manual intervention.
* **Export size limit:** Data exports (CSV downloads) are capped at **100,000 rows** per request. This limit is enforced by the EXPORT\_MAX\_ROWS configuration and CI tests (an **export-guard** step fails any build if an export would exceed this). Additionally, only **administrators** can initiate large exports from the Insights API, preventing regular users from pulling excessive data.
* **Privacy protections:** All user identifiers in the analytics database are stored as **SHA-256 hashes** (with a secret salt) instead of raw IDs. This one-way hashing ensures that personal data in analytics cannot be reverse-engineered. The ETL pipeline also **excludes small cohorts** – any aggregated result representing fewer than 10 users is dropped – to enforce *k*-anonymity and prevent re-identification of individuals in low-count data.
* **Access control & auditing:** A dedicated database role (reports\_reader) is used for read-only access to Insights data. This role is restricted to querying **dimension tables and materialized views only**; direct selects on raw fact tables require the ETL service’s privileged account. All API requests to the Insights service are logged to a central audit index (shipped via Fluent Bit to OpenSearch) with a **6 month retention** for compliance. These measures ensure sensitive analytical data is tightly governed and auditable.
* **Routing invariants:** The /reports URL namespace is reserved exclusively for the Insights module’s UI and API endpoints. Frontend routes such as /reports/smart-vote, /reports/usage, /reports/perf etc., map to the Insights dashboards, and the backend serves corresponding /api/reports/\* endpoints. This prefix has been added to the platform’s navigation map invariants, so no other module may use /reports (or its WebSocket channel /ws/reports/\*) without an official change to the reference.

## **ETL & Airflow Job Configuration**

The Insights module includes a suite of **Airflow DAGs** that perform periodic extract-transform-load (ETL) tasks and maintenance. Key jobs and their schedules are summarized below (in an etl\_config.yml style format):

etl\_dags:

- id: etl\_smart\_vote

schedule: "\*/10 \* \* \* \*" # every 10 minutes

task: "Load new votes from OLTP into smart\_vote\_fact"

- id: etl\_usage

schedule: "0 \* \* \* \*" # hourly

task: "Update usage\_mau\_fact (MAU, projects, docs counts)"

- id: etl\_perf

schedule: "\*/15 \* \* \* \*" # every 15 minutes

task: "Ingest Prometheus API metrics into api\_perf\_fact"

- id: refresh\_mat\_views

schedule: "5 \* \* \* \*" # every hour at :05

task: "REFRESH all analytics materialized views (vw\_\*)"

- id: cleanup\_cache

schedule: "@hourly" # every hour

task: "Purge Redis analytics cache keys older than 12h"

- id: purge\_old\_partitions

schedule: "0 4 \* \* 0" # every Sunday 04:00 UTC

task: "Drop any fact table partitions past retention window"

Each DAG uses Vault-managed connections for the analytics Postgres and Redis (no credentials are hard-coded). The schedules and logic align with the data retention policy – for instance, purge\_old\_partitions runs weekly to cull expired partitions. The ETL jobs ensure that analytical facts are kept up-to-date in near real-time (e.g. new votes reflected in metrics within 10 minutes) and that supporting structures (materialized views, caches) are maintained automatically.

## **Environment Variables and Secrets**

Several new environment variables (and mounted secrets) are introduced for the Insights module. These are defined following the Cookiecutter Django conventions and are split between **in-code .env settings**, **Kubernetes ConfigMaps**, and **Vault secrets** as appropriate:

| **Environment Variable** | **Used By** | **Default (Dev) / Example** | **Source & Notes** |
| --- | --- | --- | --- |
| **REPORTS\_DB\_URL** | Reports API & ETL | *none* (set per environment) | Vault secret analytics/pg/url (PostgreSQL connection string for the analytics DB, using a read-only user). In local/dev, this can point to the same Postgres with a separate schema or a dedicated analytics DB. |
| **REDIS\_URL** | Reports API & ETL | redis://redis:6379/0 | Vault secret common/redis/url. Reuses the platform’s existing Redis instance for caching (TTL as above). No new Redis env var is needed if one is already configured globally. |
| **AIRFLOW\_\_CORE\_\_FERNET\_KEY** | Airflow scheduler | *none* (generated secret) | Vault secret analytics/airflow/fernet. Used to encrypt Airflow connection creds. Set by ops (not stored in code). |
| **PROMETHEUS\_BASE\_URL** | ETL tasks | e.g. http://prometheus:9090 | ConfigMap analytics-settings. Base URL for Prometheus API (provides metrics to the etl\_perf DAG). In production, points to the cluster’s Prom service; in dev, can be left blank or pointed to a test endpoint. |
| **EXPORT\_MAX\_ROWS** | Reports API & CI | 100000 | ConfigMap analytics-settings. Max rows allowed in any export query. The backend reads this to enforce the CSV limit (and tests use it to simulate large export scenarios). |

All the above should be added to the appropriate environment config files or secret stores. For example, in local development they can reside in .env.local (or Django .envs/.local/.django), while in production the sensitive values come from **Vault** (for secrets) and a read-only **ConfigMap** (for non-sensitive settings). This separation ensures that default values are provided for development/CI, and that production deployments have the correct secure endpoints and credentials injected. No new global environment variables were introduced outside of the Insights context (existing ones like REDIS\_URL are reused), avoiding duplication of already-declared values.

## **Deployment, Scaling and Monitoring**

The Insights module is deployed as a set of dedicated services with resource limits aligned to its workload. It consists of a **reports API service**, a background **ETL worker service**, and an **Airflow** instance for orchestration (plus ancillary jobs). Key resource allocations and scaling parameters are as follows:

* **Reports API:** Deployed as a container (image ghcr.io/konnaxion/reports-api) with **3 replicas** by default. An HPA (Horizontal Pod Autoscaler) is set between **2** and **6** pods based on load. Each pod requests **300 mCPU** (0.3 CPU) and can burst up to **600 mCPU**, with **384 MiB** base RAM (limit **768 MiB**). The HPA triggers scale-up when average CPU > **60%** or when request latency P95 exceeds **400 ms** (as reported by Prometheus), ensuring the API meets its performance SLOs.
* **ETL Worker:** Deployed as reports-etl-worker (image ghcr.io/konnaxion/reports-etl) with **2 pods** (fixed) for parallel data processing. Each worker has a **400 mCPU** baseline (up to 800 mCPU) and **512 MiB** of memory (up to 1 GiB) allocated. Auto-scaling is not enabled for ETL workers (scaling is manual or via future tuning), since ETL load is relatively predictable and contained.
* **Airflow Scheduler & Workers:** The Airflow component (image apache/airflow:2.9-python3.12) runs with **1 scheduler pod** and **2 worker pods**. Each is given roughly **0.25 CPU** (250 mCPU, up to 500 mCPU) and **512 MiB** RAM (up to 1 GiB), sufficient for orchestrating the periodic jobs. The Airflow service ensures DAGs (as listed above) execute on schedule; its database (metadata DB) is a lightweight Postgres (which is included in the backup plan).
* **Database migration job:** On each deployment, a one-time migration Job (reports-db-migrate) runs to apply any analytics DB schema changes. It uses the same image as the Reports API and is given a small resource slice (approx **0.25 CPU** and **256 MiB** RAM), since migrations are usually quick. This job ensures the analytics schema (tables, partitions, views) is up-to-date before the app/ETL start running.
* **Monitoring & alerts:** Comprehensive metrics and alerting are in place for the Insights module. Custom **Prometheus** metrics track API performance (latency, error rates), ETL outcomes, and cache usage, feeding both the HPA and alerting system. For example, an alert triggers if the **95th percentile** API latency goes over **0.4 s** sustained or if the error rate exceeds **2%**. Likewise, any ETL DAG failure will raise a critical alert, and a warning is issued if a materialized view refresh exceeds **60 s**. These thresholds align with the module’s SLOs and ensure prompt attention to issues. Dashboards (e.g. *reports-api* and *reports-etl* Grafana boards) visualize throughput, cache hit ratios, and ETL runtimes. By monitoring these indicators and using the autoscaling policies above, the platform maintains the Insights service’s reliability and performance within defined limits.

Every parameter for the Insights module – from environment variables to ETL schedules, resource limits, and alert thresholds – is now **finalized** and documented, with no remaining TBD values. This extension integrates cleanly with the existing v14 reference structure, complementing the global and module-specific settings already defined. Future changes to these parameters will follow the same rigorous change-control process as the rest of the platform documentation, ensuring consistency across CI/CD and production environments. All components of the “Insights” slice are thereby production-ready, with clear defaults and governance for configuration.