Data Archival Service

Overview:

Data Archival Service for relational databases. Archive old rows from a source database into an archive database based on per-table policies, enforce RBAC for viewing archives, and run scheduled archival/purge jobs with APScheduler.

Features:

- Built with FastAPI for high performance.

- Dockerized for easy deployment.

- RBAC-protected REST API: JWT-based auth.

- Create user and assign them permission for specific tables.

- Admin has access to all tables; users can access only tables they're permitted for.

- Configure when to archive and when to delete archived data. Example: archive after 30 days, delete from archive after 365 days.

- APScheduler: In-process job scheduling for archive-delete job.

High-Level Architecture

- FastAPI app exposing:
    - Auth routes (signup/login).
    - Config routes (create/update/list policies).
    - Archive read routes.
    - System routes (health, readiness, scheduler info).
- Two databases:
    - Source DB: operational data (tables to be archived)
    - Archive DB: archived_data

- Scheduling:
    - APScheduler runs inside the API process, triggering archival and purge jobs periodically.
- Security:
    - JWT tokens include sub, role, permissions, exp.

Scheduling

- APScheduler (AsyncIOScheduler) inside API process:
    - archival_tick: periodic (e.g., every 5 minutes) runs archive_and_delete_job.
    - purge_daily: cron 02:00 UTC runs purge_expired_archives.
- Job behavior:
    - Archival: for each policy, select rows older than created_at < now - archive_after_days (AND custom_criteria if present), copy payload to archive as JSON, commit, then delete source rows by id.
    - Purge: for each policy, delete archive rows with archived_at < now - delete_after_days.
- **Key decision:** Use APScheduler for simplicity in single-instance deployments.

API Design

- Auth:
    - POST /auth/signup: create regular user (role enforced server-side as “user”).
    - POST /auth/login: returns access_token and token_type.
- Config:
    - POST /config: create/update policy for a table (admin-only).
    - GET /config/: list all policies (admin-only).
    - GET /config/{table_name}: view single policy (admin or table manager).

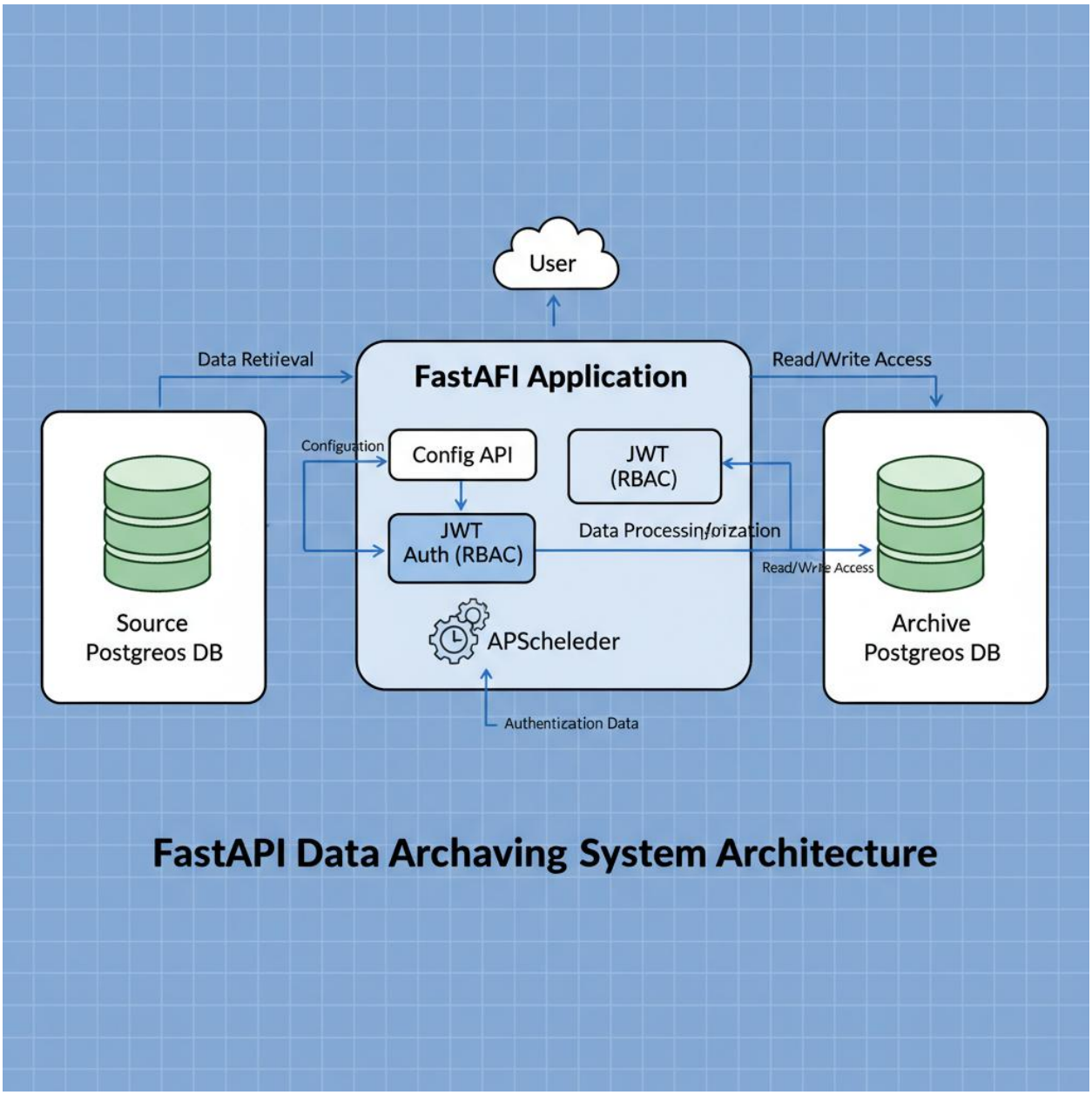
- Archives:
    - GET /archives/{table_name}: list archived items.
- System:
    - GET /health: liveness.
    - GET /ready: DB readiness
    - GET /system/scheduler/jobs: list jobs and next_run_time (observe scheduler).
    - POST /system/scheduler/test: schedule a simple test job (for verification).

Key Decisions:

1. Two databases (Source/Archive):
    - Avoid coupling schemas and enable independent lifecycle of control-plane and archive store.
2. Scheduling (APScheduler):
    - Simplifies ops for single-instance deployments; acceptable for modest workloads.
3. JWT with embedded permissions:
    - Fast, stateless checks; accept re-login requirement after permission changes.
4. Admin/table manager separation:
    - Separated permissions based on role, admin can do/view all

Note: Please read README file in code repository for more information related to quick start guide.

Architectural Diagram:



FastAPI Data Archaving System Architecture