

RePan: Regional Pandemic Analytics

Architecture & Developer Documentation

1. Platform Scope

RePan is a containerized analytics platform that aggregates regional health data, applies processing pipelines, and exposes curated datasets through dashboards. Its architecture is built around a cloud-native data lake using object storage (MinIO), Parquet files, and schema-on-read querying by Apache Druid. User-facing services include a Django backend, a Next.js frontend, and Apache Superset for dashboarding. Workflow orchestration is handled by Apache Airflow, while Apache Hop performs ETL tasks.

2. Data Lake Principles in RePan

RePan's architecture implements core data lake concepts:

1. Centralized Object Storage

- MinIO provides S3-compatible storage. Buckets are provisioned for raw pipelines (pipelines/), processed datasets (parquets/), and user assets (avatars/).
- Storage is decoupled from compute; data is accessible to Hop, Airflow, Druid, and Superset via S3 endpoints.

2. Schema-on-Read & Parquet Format

- ETL pipelines generate partitioned Parquet files, preserving raw column names and types.
- Druid ingests these Parquet files with its parquet extension, enabling schema discovery at query time.

3. Separation of Zones

- **Raw Zone:** Pipeline definitions and source data stored under pipelines/.
- **Staging Zone:** Parquet outputs generated by Hop are uploaded to parquets/<pipeline>/<run_id>/.
- **Curated Zone:** Druid segments created from Parquet files serve analytics queries. Superset catalogs these datasets for visualization.

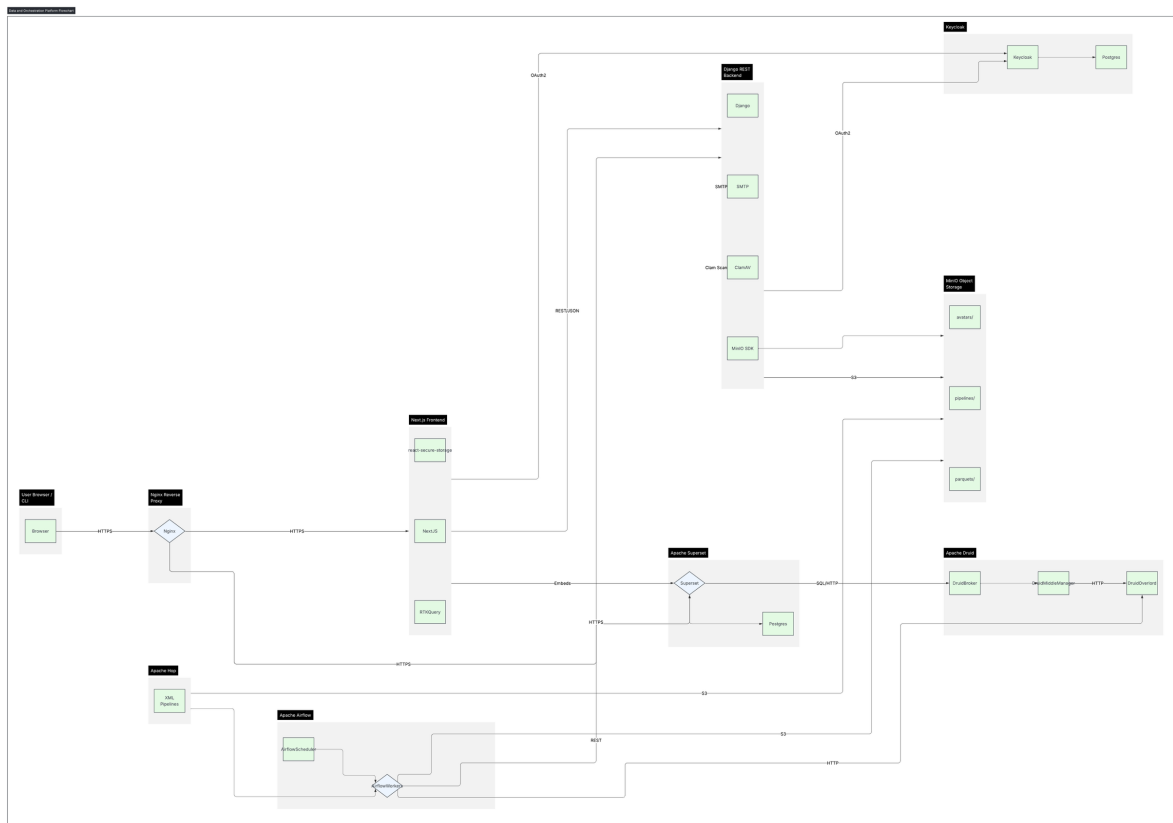
4. Metadata & Governance

- Airflow DAGs capture lineage: each ingestion run references pipeline files and output paths.
- Superset maintains dataset metadata (column types, metrics, dashboards).
- Keycloak enforces role-based access to datasets and APIs.

5. Audit & Versioning

- Each Hop run writes timestamped Parquet files; historical runs remain accessible.
- Airflow logs and Superset's dataset versioning track data evolution.

3. Detailed Architecture Diagram



4. Component Architecture

4.1 Authentication & Authorization (Keycloak)

- Realms define clients for backend, frontend, and Superset.
- Roles (realm and client) govern API access and dashboard visibility.
- Frontend and backend rely on OAuth2 flows; access tokens carry role claims.
- Admin operations use Keycloak's admin REST API to manage users, roles, and passwords.

4.2 Backend (Django)

- **Frameworks & Libraries:** Django, Django REST Framework, drf_yasg, minio, pyclamd, python-keycloak.
- **Key Apps**
 - **accounts:** User CRUD, role assignment, avatar upload.
 - **auth:** Session endpoints, token management, password reset/change.
 - **pipeline:** Validates Hop XML, stores pipeline files, tracks Parquet output paths.
 - **process:** Manages Airflow DAG definitions and run history.
 - **supersets:** Fetches dashboards, charts, favorites, guest tokens.
- **Data Lake Integration**
 - Pipeline uploads: Users send XML pipelines to backend; backend scans via ClamAV and stores under `pipelines/`.
 - Pipeline validation: `ParquetFileOutput` steps are checked for correct filename extensions and disabled date suffixes.
 - Airflow triggers: Backend writes DAG definitions referencing pipeline paths and Parquet destinations.
- **User Management Flow**

1. API call authenticated via Keycloak token.
2. `has_admin_role` verifies roles.
3. `UserListView` or `UserDetailView` interacts with Keycloak admin API.
4. New users receive temporary passwords via email; profile updates allow avatar upload to MinIO.

4.3 Frontend (Next.js)

- **Frameworks & Libraries:** React, Next.js, Redux Toolkit, RTK Query, Tailwind CSS, react-secure-storage.
- **Modules**
 - `auth`: Login/logout via OAuth code flow; tokens stored in secure storage.
 - `user`: Profile CRUD, avatar upload, password reset.
 - `pipeline & process`: Upload, list, delete pipelines; monitor Airflow DAGs and runs.
 - `superset`: Fetch dashboards/charts, guest tokens, favorites.
- **Data Lake Integration**
 - Users upload pipelines through the frontend; responses include S3 locations.
 - Dataset metadata retrieved from backend for display and Superset embedding.

4.4 ETL Processing (Apache Hop)

- Pipelines reside in `hop/pipelines`. They contain `ParquetFileOutput` steps with explicit file extensions (`.parquet`) and disabled date/time suffixes.
- Hop runs within a container; Airflow tasks execute Hop pipelines, writing Parquet files to MinIO.
- Parameterization allows pipelines to adapt to environment variables for S3 paths.

4.5 Workflow Orchestration (Apache Airflow)

- Custom DAGs download pipelines from MinIO (`download_pipeline_minio`), adjust paths, and execute Hop jobs.
- After pipeline execution, a `DruidOperator` reads the schema of the generated Parquet file, constructs an ingestion spec, and instructs Druid to batch ingest the dataset.
- Superset is notified via API to create or update dataset metadata, enabling immediate visualization.

4.6 Analytics Database (Apache Druid)

- Configured with `druid-parquet-extensions` for native Parquet ingestion.
- Middle Managers handle batch ingestion tasks, referencing Parquet files from MinIO via S3 paths.
- Brokers expose SQL/HTTP endpoints for query access by Superset.
- Segments are stored in deep storage (S3-compatible) and made available for queries; Druid handles time-based partitions and roll-ups.

4.7 Business Intelligence (Apache Superset)

- Connected to Druid via SQLAlchemy; uses Keycloak OIDC for authentication.
- Dashboards and charts reference Druid datasets created during Airflow ingestion.
- Guest tokens support iframe embedding in the frontend without requiring full login.
- Metadata stored in Superset's own Postgres database tracks datasets, slices, and dashboards.

4.8 Reverse Proxy & Security (Nginx + ClamAV)

- Nginx terminates TLS and routes requests to backend, frontend, Superset, and Druid.
- ClamAV runs as a service; backend scans uploaded files through `pyclamd` before storing in MinIO.

5. Data Flow

1. **Pipeline Upload**
 - Developer uploads a Hop pipeline via frontend; backend scans and stores it in `pipelines/`.
2. **Pipeline Validation**
 - Backend validates `ParquetFileOutput` options, ensuring `.parquet` extension and disabled suffixes.
3. **Airflow Execution**
 - Airflow downloads the pipeline from MinIO, updates output paths to `parquets/<pipeline>/<run_id>/`, and invokes Hop.
4. **Parquet Generation**
 - Hop writes Parquet files directly to MinIO; files retain original schema and support partitioning.
5. **Druid Ingestion**
 - Airflow constructs a Druid ingestion spec referencing the Parquet file S3 path; Druid ingests data into a new or existing datasource.
6. **Dataset Registration**
 - Upon successful ingestion, Airflow calls Superset to create or update the dataset, including field types and metrics.
7. **Visualization**
 - Frontend fetches Superset guest tokens and embeds dashboards; users explore datasets with filter and chart interactions.

6. Deployment & Operations

6.1 Local Development

- Scripts: `add-local-hosts.sh` maps service hostnames; `gen-local-certs.sh` generates local TLS certificates.
- `make start-local` spins up the entire stack; `make start-local service=<name>` targets specific services.
- Environment variables define connections among services (e.g., `MINIO_ENDPOINT`, `DRUID_BROKER_URL`).

6.2 Production Considerations

- Use persistent volumes for MinIO and databases.
- Secure Keycloak with HTTPS and proper realm settings.
- Configure Druid deep storage and metadata storage.
- Implement backups for MinIO buckets and Postgres databases.
- Employ monitoring (Prometheus, Grafana) for service health and ingestion metrics.

7. COHIS (Cameroon) Branch

The COHIS branch extends RePan for Cameroon's health system. While not included in this repository snapshot, typical adaptations include:

1. **Pipelines & Data Sources**
 - Additional Hop pipelines ingesting data from Cameroon's health information systems (e.g., DHIS2, SORMAS).
 - Custom transformation logic for local indicators and reporting requirements.
2. **Localization**
 - Frontend translations into French and English.

- Default country codes, date formats, and health metrics tailored to Cameroon.
3. **Environment & Deployment**
 - Different hostnames and certificates (*.cohis.local).
 - Keycloak realm tuned to Cameroon's governance model, with role definitions aligning to local policies.
 4. **Governance & Access**
 - Role mappings in Keycloak for national, regional, and facility-level access.
 - Superset dashboards preconfigured with Cameroon-specific datasets and visuals.
 5. **Additional Data Lake Practices**
 - Dedicated MinIO buckets for Cameroon to segregate raw and curated data.
 - Airflow DAGs reflecting local ingestion schedules and data retention policies.

Developers implementing COHIS should branch from `main`, introduce Cameroon-specific configurations across backend, frontend, and infrastructure, and ensure all pipelines align with the data lake standards established in RePan.

8. Contribution Guidelines

- Adhere to existing Django app structure and Next.js modular architecture.
- Validate any new Hop pipelines for correct Parquet settings.
- Include unit tests for backend and frontend modules.
- Document new APIs using Swagger annotations.
- Ensure new services or pipelines integrate with Keycloak for access control.
- Perform security checks on uploads and maintain consistent bucket naming conventions.