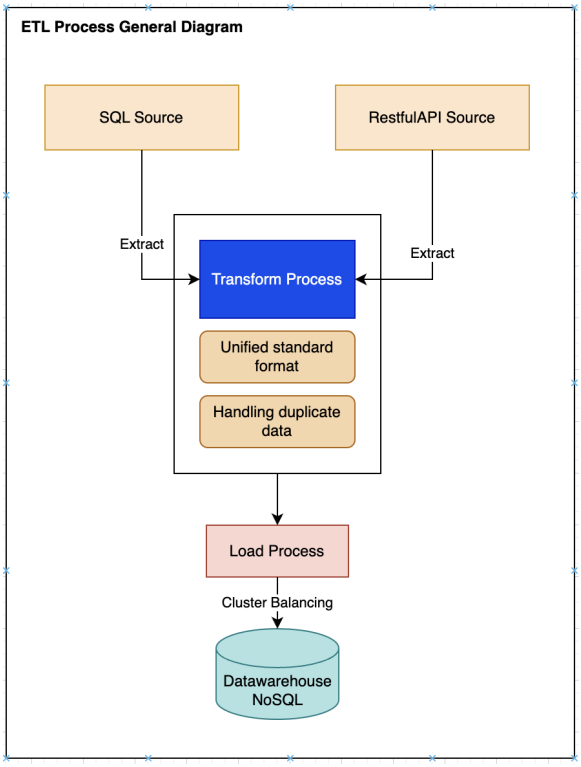
Giả sử bạn đang làm việc trong một dự án Customer Data Platform và được giao nhiệm vụ tích hợp dữ liệu khách hàng từ hai nguồn khác nhau: một cơ sở dữ liệu SQL và một API RESTful. Dữ liệu từ hai nguồn này cần được hợp nhất thành một định dạng duy nhất để lưu trữ trong một cơ sở dữ liệu NoSQL. Hãy mô tả cách bạn sẽ thực hiện nhiệm vụ này, bao gồm các công cụ và công nghệ bạn sẽ sử dụng, và cách bạn đảm bảo tính bảo mật của dữ liệu trong quá trình tích hợp.

Giải pháp sử dụng quy trình ETL để xử lý tổng hợp Data khách hàng.



✺ Xác định các khó khăn sẽ gặp trong quy trình ETL.

- **Sai khác về cấu trúc dữ liệu giữa các nguồn dữ liệu (Formatting):** Đồng bộ cấu trúc dữ liệu là thao tác sẽ gặp nhiều khó khăn nhất, bởi vì dữ liệu từ SQL sẽ trả về cấu trúc bảng, còn Restful API thông thường sẽ trả về JSON/XML. Vì vậy cần xác nhận chuẩn dữ liệu sẽ sử dụng để thực hiện các thao tác tiếp theo. Trong ví dụ xử lý bài toán này tôi dùng **JSON** để xử lý.

- **Dữ liệu bị trùng lặp và mâu thuẫn dữ liệu (Duplicates), Độ ưu tiên của nguồn dữ liệu (Priority)**: Trong quá trình xử lý dữ liệu, sẽ có rất nhiều trường hợp khách hàng xuất hiện ở cả 2 nguồn dữ liệu, cần xử lý theo độ ưu tiên dữ liệu theo nguồn được chọn ưu tiên, và bổ sung dữ liệu cho nhau khi xảy ra vấn đề bị thiếu dữ liệu, đảm bảo dữ liệu cuối cùng đầy đủ tiêu chuẩn so với ban đầu đặt ra.

- **Xác định khối lượng và tần xuất cần xử lý**: ETL thông thường được xây dựng để xử lý một lượng dữ liệu nhất định sau khi các nguồn phát sinh dữ liệu, vì vậy thiết lập plan và số lượng dữ liệu cần chạy cũng là một thách thức cần phải đảm bảo.

✺ Quy trình thực hiện ETL tích hợp dữ liệu khách hàng

Extract (Trích xuất dữ liệu)

- Kết nối đến cơ sở dữ liệu SQL qua driver phù hợp (JDBC, ODBC, v.v).
- Gọi API RESTful sử dụng HTTP client (RestTemplate, Axios, hoặc curl).
- Trích xuất dữ liệu theo định kỳ hoặc theo sự kiện (event-driven), đảm bảo lấy đủ dữ liệu mới hoặc thay đổi (incremental load).

- Định dạng dữ liệu trích xuất từ SQL thành JSON đồng bộ với định dạng API trả về để dễ xử lý bước sau.

Transform (Chuyển đổi dữ liệu)

- Chuẩn hóa cấu trúc dữ liệu, chuyển đổi các trường, tên trường sao cho thống nhất.
- Xử lý dữ liệu trùng lặp: Áp dụng quy tắc ưu tiên nguồn dữ liệu chính, gộp dữ liệu từ nguồn phụ nếu có thiếu sót.
- Làm sạch dữ liệu: Loại bỏ bản ghi lỗi, xử lý giá trị null, chuẩn hóa định dạng ngày tháng, số điện thoại, email.
- Ánh xạ dữ liệu: Chuẩn hóa các trường enum, trạng thái, phân loại theo chuẩn nội bộ.
- Tích hợp dữ liệu bổ sung nếu có từ các nguồn khác để tăng độ đầy đủ.

Load (Nạp / Lưu dữ liệu)

- Lựa chọn NoSQL database phù hợp: MongoDB, Cassandra, hoặc Elasticsearch tùy mục đích truy vấn và lưu trữ, trong trường hợp này MongoDB là lựa chọn phổ biến nhất.
- Thiết kế schema NoSQL linh hoạt, phục vụ truy vấn nhanh và mở rộng dễ dàng.
- Nạp dữ liệu đã chuẩn hóa vào NoSQL bằng batch hoặc streaming để giảm tải hệ thống.
- Đảm bảo ghi đè hoặc cập nhật đúng bản ghi khách hàng theo khóa chính (ID duy nhất).

✺ Công cụ và công nghệ sử dụng

Ngôn ngữ và Framework (Recommend)

- Node.js (v18+): Xử lý JSON tự nhiên, hiệu quả, async/await đơn giản, hổ trợ typescript để đảm bảo tránh phát sinh lỗi về kiểu dữ liệu khi xử lý, Hệ sinh

thái thư viện phong phú, phù hợp để đáp ứng các quy trình ETL nhỏ và trung bình.

Công cụ trích xuất dữ liệu

- **node-postgres (pg)** hoặc driver tương ứng cho SQL database.
- **Axios** để gọi API RESTful qua HTTPS.

Công cụ xử lý và chuyển đổi dữ liệu

- Node.js native JSON và thư viện **ajv** để validate schema JSON.

Công cụ lưu trữ NoSQL

- **MongoDB** (phiên bản mới nhất).
- Driver chính thức **mongodb** hoặc ORM **Mongoose**.

Công cụ quản lý configs và secrects

- **dotenv** để quản lý biến môi trường.
- **AWS Secrets Manager** kết hợp SDK Node.js lấy secrets động.

Công cụ scheduling và workflow

- **node-cron** để lập lịch chạy các job định kỳ.

Công cụ logging và monitoring

- **Winston** hoặc **Pino** cho logging có cấu hình và có thể kết hợp với **Kibana ELK** để đẩy log lên UI theo dõi.

✺ Biện pháp bảo mật dữ liệu trong ETL

Bảo mật kết nối

- Kết nối SQL database và MongoDB qua **TLS/SSL** để mã hóa dữ liệu khi truyền.
- API RESTful phải gọi qua **HTTPS** với chứng chỉ hợp lệ và còn thời hạn.
- Kiểm tra nguồn kết nối, lấy dữ liệu, thiết lập CORS (quy định kết nối, method kết nối)
- Đảm bảo thiết lập Whitelist IP được phép truy cập vào quy trình.

Xác thực và phân quyền

- Sử dụng **OAuth2** hoặc **API key** có hạn cho API.
- MongoDB phân quyền theo **RBAC**, tạo user chỉ có quyền cần thiết.
- Có hệ thống Backend quản lý tài khoản truy cập quy trình ETL

Quản lý configs và secrets

- Không lưu mật khẩu, token trong mã nguồn (hard code).
- Sử dụng **AWS Secrets Manager** như đã đề cập ở trên.

Mã hóa dữ liệu nhạy cảm (Sentitive data)

- Mã hóa dữ liệu cá nhân (email, điện thoại) trước khi lưu MongoDB.
- Kích hoạt **Encryption at Rest** nếu sử dụng **MongoDB Atlas** hoặc phiên bản **Enterprise**.
- Cơ chế **Decrypt** dữ liệu tại Backend có thể bổ sung quy trình sử dụng **HSM**

Giám sát và audit

- Ghi log truy cập, thay đổi dữ liệu chi tiết.
- Thiết lập cảnh báo tự động khi phát hiện lỗi hoặc truy cập bất thường.

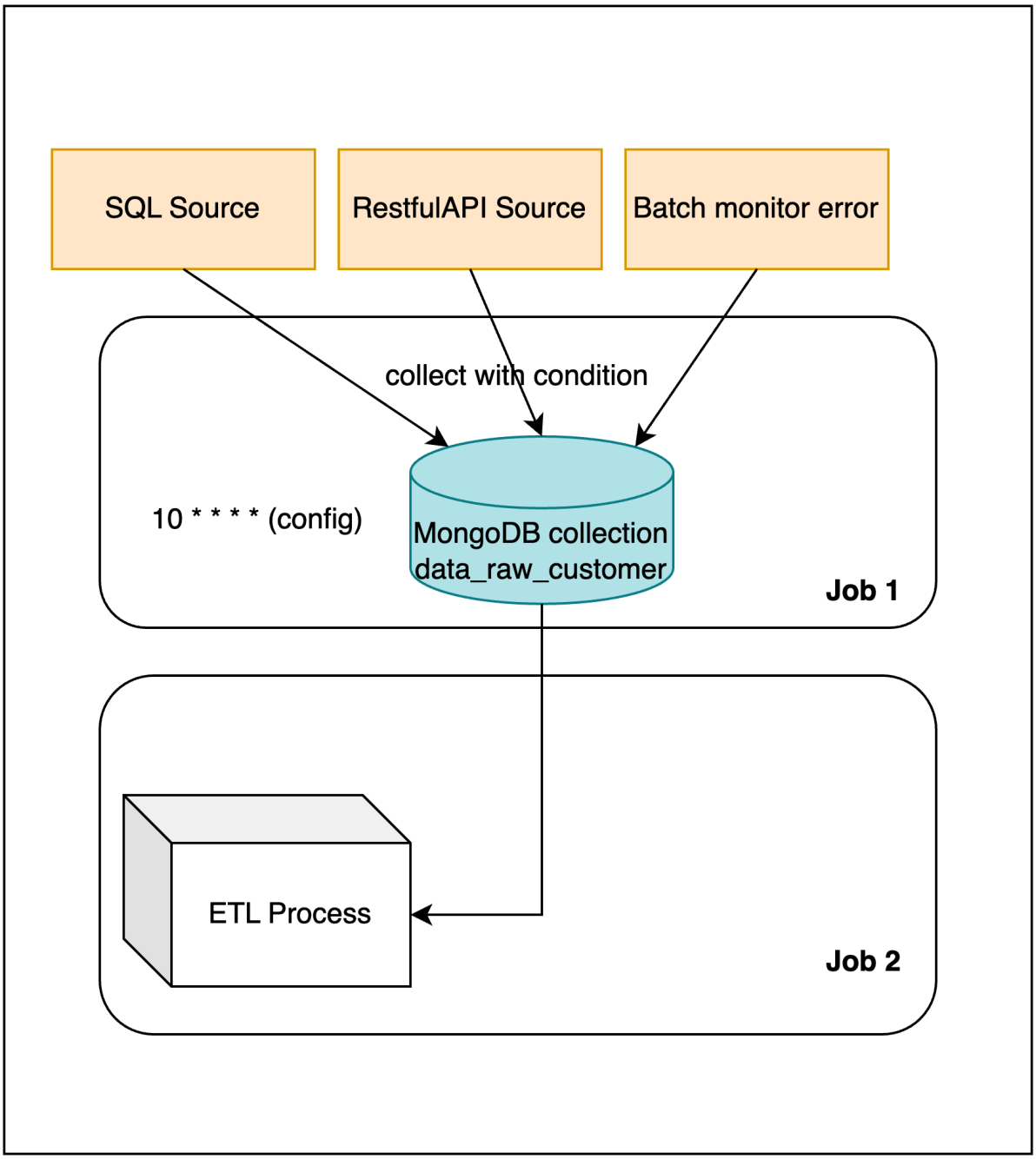
✺ Triển khai hệ thống ETL

Kiến trúc triển khai hệ thống ETL đề nghị, tôi sẽ xây dựng 2 job với vai trò khác nhau.

Job 1: Job này sẽ thực hiện collect dữ liệu theo một điều kiện nhất định (ví dụ 10000 dữ liệu mới nhất, 5000 dữ liệu ở dữ liệu city là TP.Hồ Chi Minh ..), hoặc lấy các dữ liệu mà quá trình ETL không thực hiện được (Batch error record), sau đó lưu trữ tạm vào MongoDB collection (Ở đây có thể tuỳ chọn lưu file, hoặc các khác tuỳ ý).

Job 2: Job này sẽ lấy các dữ liệu từ Job 1 tổng hợp trước đó, và đưa vào luồng xử lý của ETL.

Cơ chế **Retry** và **Batch Monitor** sẽ được tích hợp để đảm bảo xử lý ổn định và dễ phục hồi khi gặp lỗi.



Cơ chế Retry - Áp dụng cho cả 2 job, đặc biệt các thao tác gọi API, truy vấn SQL, và ghi dữ liệu MongoDB.

Retry theo nguyên tắc:

- Giới hạn số lần retry (thường 3-5 lần).
- Delay tăng dần giữa các lần (exponential backoff).
- Retry chỉ với lỗi tạm thời (network timeout, deadlock, service unavailable).

Khi hết retry vẫn lỗi, thông tin lỗi sẽ được ghi xuống **Batch Monitor** để xử lý tiếp vào luồng khác.
Ví dụ coding cho retry (Node.js, sử dụng package **promise-retry**):

```
import promiseRetry from 'promise-retry';

async function callApiWithRetry(url, options) {
 return promiseRetry((retry, number) => {
   console.log(`API call attempt #${number}`);
   return axios.get(url, options).catch(err => {
     if (isTransientError(err)) {
       retry(err);
     }
     throw err;
   });
 }, { retries: 3, minTimeout: 1000, factor: 2 });
}
```

Cơ chế Batch Monitor - Xử lý lỗi và ghi nhận - Khi xử lý một bản ghi hoặc batch gặp lỗi không khắc phục được ngay, lưu toàn bộ dữ liệu và thông tin lỗi vào collection batch_monitor (tuỳ ý) sau đó xử lý lại bằng Job định kỳ, hoặc như đề cập có thể dùng Job 1 để lấy dữ liệu từ batch_monitor để xử lý lại.

Tạo một collection để lưu các record lỗi, đề nghị như sau

Trường	Mô tả
_id	ObjectId MongoDB tự sinh
batch_data	Dữ liệu lỗi (bản ghi hoặc batch JSON)
error_message	Mô tả lỗi chi tiết
error_res_code	Phân loại lỗi (validation, hệ thống, kết nối)

timestamp	Thời điểm lỗi xảy ra
status	pending / retrying / done / failed …
retry_count	Số lần đã retry
last_retry_time	Thời gian retry lần gần nhất

Tạo một function để xử lý lưu trữ Batch

```
async function saveErrorBatch(db, record, errorMessage, errorType = 'system') {
 const monitorCol = db.collection('batch_monitor');

 await monitorCol.insertOne({
   batch_data: record,
   error_message: errorMessage,
   error_type: errorType,
   timestamp: new Date(),
   status: 'pending',
   retry_count: 0,
 });

 console.log(`Saved error batch: ${errorMessage}`);
}
```

Cơ chế Locking Database - Sử dụng upsert atomic cho MongoDB, tránh deadlock.

Một đoạn ví dụ triển khai với upsert atomic trong MongoDB

```
import { MongoClient } from 'mongodb';

async function upsertCustomerRecord(record) {
 const client = new MongoClient(process.env.MONGO_URI, { tls: true });
 await client.connect();

 const db = client.db(process.env.MONGO_DB);
 const collection = db.collection('customers');
```

```
    // Upsert atomic theo id
    await collection.replaceOne({ id: record.id }, record, { upsert: true });

    console.log(`Upserted customer id=${record.id}`);

    await client.close();
}
```

Ngoài các cơ chế trên, hệ thống có thể cần đánh giá thêm về chiến lược **Incremental Load** (Đánh dấu điểm ghi nhận thành công vào hệ thống NoSQL) - chiến lược **Validation và Chất lượng Dữ liệu** (Data Quality) - kiểm tra các giá trị theo chuẩn quy định của các tổ chức, ví dụ như email, phone number, date of birth, id.

Mã nguồn tham khảo: https://github.com/nhockool1002/ETL-sample-project/tree/main/local

✳ Giải pháp bổ sung ETL với AWS Cloud (tìm hiểu)

Kiến trúc tổng quan trên AWS

Dữ liệu nguồn:

- Cơ sở dữ liệu SQL (ví dụ RDS PostgreSQL, MySQL).
- API RESTful bên ngoài hoặc nội bộ.

Dịch vụ AWS sử dụng:

- **AWS Glue:** dịch vụ ETL serverless tự động trích xuất, chuyển đổi, tải dữ liệu, hỗ trợ catalog dữ liệu.
- **Amazon S3:** lưu trữ tạm hoặc kết quả ETL dưới dạng file (JSON, Parquet).
- **Amazon DynamoDB / Amazon DocumentDB:** thay thế hoặc bổ sung cho MongoDB làm NoSQL database.
- **AWS Lambda:** chạy các job nhỏ, trigger event-driven.
- **Amazon EventBridge / CloudWatch Events:** lên lịch chạy job định kỳ.
- **AWS Secrets Manager:** quản lý và phân phối secrets (mật khẩu DB, token API) an toàn.
- **AWS KMS (Key Management Service):** mã hóa dữ liệu nhạy cảm, khóa mã hóa tích hợp với Glue, Lambda.

- **Amazon CloudWatch:** giám sát, logging, cảnh báo tự động.

Quy trình ETL trên AWS

Extract:

- Glue sử dụng JDBC connection lấy dữ liệu từ RDS (SQL).
- Lambda hoặc Glue lấy dữ liệu từ API RESTful (gọi HTTPS với token lấy từ Secrets Manager).
- Dữ liệu tạm lưu S3, giúp xử lý batch lớn, đồng bộ dữ liệu.

Transform:

- Glue sử dụng Apache Spark xử lý dữ liệu lớn, chuẩn hóa dữ liệu, validate và làm sạch.
- Mã hóa dữ liệu nhạy cảm sử dụng KMS trực tiếp trong Glue hoặc Lambda.
- Ánh xạ, hợp nhất dữ liệu từ nhiều nguồn.

Load:

- Dữ liệu nạp vào DynamoDB hoặc DocumentDB, tận dụng các tính năng NoSQL nhanh, mở rộng cao.
- Hoặc nạp trở lại S3 cho các hệ thống downstream phân tích.

Bảo mật và quản lý secrets

- AWS Secrets Manager quản lý mật khẩu DB, token API RESTful, được Lambda/Glue gọi lấy secrets động, không lưu trong mã nguồn hay biến môi trường.
- KMS dùng để mã hóa dữ liệu nhạy cảm trong khi lưu trữ hoặc truyền tải.
- Giới hạn truy cập mạng bằng VPC Endpoint cho Glue, Lambda với RDS, DocumentDB, S3.
- Sử dụng IAM Role với quyền tối thiểu cho từng dịch vụ.

Lên lịch và giám sát

- Dùng EventBridge / CloudWatch Events để lên lịch tự động chạy job Glue hoặc Lambda thay cho node-cron.
- CloudWatch Logs lưu toàn bộ log, cảnh báo lỗi, cảnh báo bất thường (ví dụ số lần retry vượt mức).
- Dùng AWS X-Ray để theo dõi luồng xử lý, phân tích bottleneck.

Cơ chế retry và batch monitor trên AWS

- Sử dụng **AWS Step Functions** để xây dựng workflow ETL có điều kiện retry, lưu trạng thái batch lỗi.
- Batch lỗi có thể lưu trên S3 hoặc DynamoDB với trạng thái, được Lambda định kỳ đọc và xử lý lại.
- Các service AWS như Glue/Lambda tích hợp native retry policy và Dead Letter Queue (DLQ) qua SNS/SQS.

Ưu điểm khi sử dụng AWS Cloud

- Giảm thiểu quản lý hạ tầng, nâng cao khả năng mở rộng tự động theo khối lượng dữ liệu.
- Tích hợp sẵn các giải pháp bảo mật, quản lý secrets và giám sát.
- Tận dụng được các công cụ big data (Spark trên Glue) để xử lý dữ liệu lớn hiệu quả.
- Dễ dàng tích hợp với các hệ thống downstream hoặc analytics trên AWS.

Mã nguồn tham khảo: https://github.com/nhockool1002/ETL-sample-project/tree/main/aws