EDD: Data Ingestion & Adapter Strategy

Lead Backend Engineer

January 2026

Contents

1 Problem Definition

Kushim must ingest data from N different sources, each with unique schemas, authentication methods, and rate limits. Scaling this linearly without the code becoming unmaintainable requires a strict architectural pattern.

2 Proposed Solution: Modular Adapter Pattern

We define a TypeScript interface that all adapters must satisfy. This ensures the Core Service can treat every source as a black box.

2.1 The Adapter Interface

```
interface IKushimAdapter {
  providerName: string;
  fetch(config: AdapterConfig): Promise<RawData>;
  normalize(data: RawData): KushimStandardRecord[];
  validate(data: any): boolean;
}
```

3 Asynchronous Processing Pipeline

To ensure the system remains responsive under heavy load, we implement a producer-consumer model using **BullMQ**.

1. **Producer:** API receives a sync request and adds a job to `ingestion-queue`.
2. **Consumer:** Dedicated workers pull jobs. If a job fails due to network issues, it utilizes an **Exponential Backoff** retry strategy.

4 Data Integrity & Deduplication

To prevent redundant data entry, we calculate a **Content Checksum** (SHA-256) for every normalized record.

$$Checksum = H(UserID + SourceID + ExternalID + Payload) \tag{1}$$

Before insertion, the database performs a `UPSERT` based on this checksum.

5 Performance Constraints

- **Memory:** Workers must not exceed 512MB RAM per thread.
- **Latency:** Normalization logic must complete in $< 50ms$ per 100 records.