Core Data Pipeline

Product Requirements Document (PRD)

Executive Summary

A robust, automated data pipeline that ingests cryptocurrency news from RSS feeds, eliminates duplicates, and stores clean, structured content in a database. This forms the foundational infrastructure for the crypto newsletter system.

---

1. Product Overview

Vision

Create a reliable, scalable data pipeline that consistently delivers clean, deduplicated cryptocurrency news content as the foundation for downstream analysis and newsletter generation.

Core Value Proposition

- **Reliable Ingestion**: Consistent RSS feed processing every 4 hours
- **Zero Duplicates**: Intelligent deduplication across multiple sources
- **Clean Data**: Structured, normalized content ready for analysis
- **Fault Tolerance**: Robust error handling and recovery mechanisms
- **Monitoring Ready**: Full observability for system health and performance

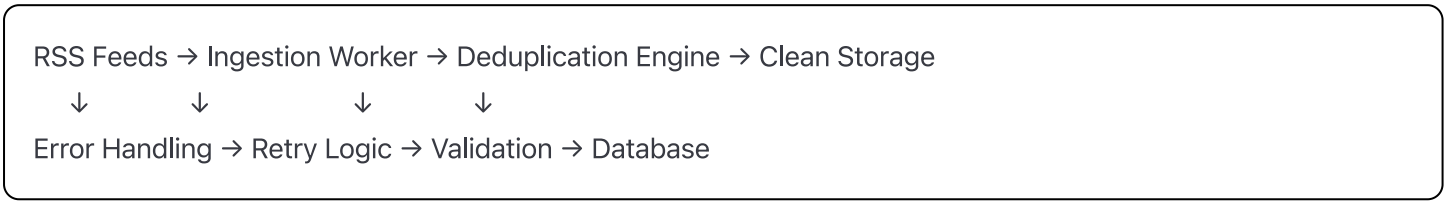
---

2. System Architecture

Infrastructure Stack

- **Task Scheduling**: Celery Beat (cron-like scheduling)
- **Task Queue**: Redis (job queue and caching)
- **Database**: Neon PostgreSQL
- **Deployment**: Railway
- **Language**: Python 3.11+

Data Flow



---

3. Functional Requirements

3.1 RSS Feed Management

Responsibilities:

- Maintain list of configured RSS feeds
- Handle feed discovery and validation
- Support feed status monitoring (active/inactive)
- Track feed reliability and performance metrics

Requirements:

- Support 10-50 RSS feeds initially
- Validate feed URLs and format compatibility
- Handle feeds with different update frequencies
- Store feed metadata and performance history

Success Criteria:

- 99% feed availability detection accuracy
- <2 second feed validation time
- Support for all major RSS/Atom formats

3.2 Content Ingestion (Every 4 Hours)

Responsibilities:

- Fetch content from all active RSS feeds
- Parse and extract article metadata
- Normalize content structure
- Handle network failures and timeouts

Requirements:

- Process all feeds within 10-minute window
- Extract: title, URL, content, publish date, author, description
- Handle various RSS formats (RSS 2.0, Atom, custom formats)
- Implement exponential backoff for failed requests
- Log all ingestion activities for debugging

Success Criteria:

- 95% successful ingestion rate per cycle
- Complete ingestion cycle within 10 minutes
- Zero data corruption during extraction
- Graceful handling of malformed feeds

3.3 Duplicate Detection & Prevention

Responsibilities:

- Identify duplicate articles across sources
- Handle republished content with minor variations
- Maintain duplicate detection accuracy over time

Requirements:

- **URL-based deduplication**: Normalize URLs (remove tracking params, fragments)
- **Content similarity**: Use title + content hash for near-duplicate detection
- **Cross-publisher detection**: Identify same story from different sources
- **Temporal awareness**: Handle delayed publications and syndication
- **Performance optimization**: Fast lookup using database indexes

Success Criteria:

- Zero duplicate articles in final dataset
- <1 second duplicate detection per article
- 99.5% accuracy in identifying true duplicates
- <0.1% false positive rate (blocking unique content)

3.4 Data Storage & Management

Responsibilities:

- Store clean, structured article data
- Maintain data integrity and relationships
- Support efficient querying for downstream processes
- Handle database migrations and schema updates

Requirements:

- Atomic transactions for data consistency
- Foreign key relationships maintained
- Efficient indexing for common queries

- Data retention policies and archiving
- Backup and recovery procedures

Success Criteria:

- 100% data integrity maintained
- Query response times <100ms for standard operations
- Database uptime >99.9%
- Successful automated backups

---

4. Technical Specifications

4.1 Database Schema

sql

```
-- Core Tables
publishers (
    id BIGSERIAL PRIMARY KEY,
    name TEXT NOT NULL,
    rss_url TEXT NOT NULL UNIQUE,
    status TEXT DEFAULT 'ACTIVE' CHECK (status IN ('ACTIVE', 'INACTIVE')),
    last_successful_fetch TIMESTAMPTZ,
    error_count INTEGER DEFAULT 0,
    created_at TIMESTAMPTZ DEFAULT NOW(),
    updated_at TIMESTAMPTZ DEFAULT NOW()
);

articles (
    id BIGSERIAL PRIMARY KEY,
    external_id BIGINT NOT NULL UNIQUE,
    guid TEXT NOT NULL UNIQUE,
    title TEXT NOT NULL,
    subtitle TEXT,
    authors TEXT,
    url TEXT NOT NULL UNIQUE,
    body TEXT,
    keywords TEXT,
    language TEXT,
    image_url TEXT,
    published_on TIMESTAMPTZ,
    publisher_id BIGINT REFERENCES publishers(id),
    status TEXT DEFAULT 'ACTIVE' CHECK (status IN ('ACTIVE', 'INACTIVE', 'DELETED')),
    created_at TIMESTAMPTZ DEFAULT NOW(),
    updated_at TIMESTAMPTZ DEFAULT NOW()
);

-- Processing Tables
ingestion_jobs (
    id BIGSERIAL PRIMARY KEY,
    job_type TEXT NOT NULL,
    status TEXT NOT NULL CHECK (status IN ('PENDING', 'RUNNING', 'COMPLETED', 'FAILED')),
    started_at TIMESTAMPTZ,
    completed_at TIMESTAMPTZ,
    error_details JSONB,
    articles_processed INTEGER DEFAULT 0,
    duplicates_found INTEGER DEFAULT 0,
    created_at TIMESTAMPTZ DEFAULT NOW()
);

feed_performance (
    id BIGSERIAL PRIMARY KEY,
```

```
    publisher_id BIGINT REFERENCES publishers(id),
    fetch_time TIMESTAMPTZ NOT NULL,
    response_time_ms INTEGER,
    articles_found INTEGER,
    status TEXT NOT NULL,
    error_message TEXT,
    created_at TIMESTAMPTZ DEFAULT NOW()
);
```

4.2 Indexing Strategy

sql

```sql
-- Performance Indexes
CREATE INDEX idx_articles_url ON articles(url);
CREATE INDEX idx_articles_guid ON articles(guid);
CREATE INDEX idx_articles_published_on ON articles(published_on);
CREATE INDEX idx_articles_publisher_id ON articles(publisher_id);
CREATE INDEX idx_articles_status ON articles(status);

-- Deduplication Indexes
CREATE INDEX idx_articles_title_hash ON articles(md5(title));
CREATE INDEX idx_articles_content_hash ON articles(md5(substring(body, 1, 1000)));

-- Search Indexes (for future use)
CREATE INDEX idx_articles_title_fts ON articles USING gin(to_tsvector('english', title));
```

4.3 Celery Configuration

python

```python
class CeleryConfig:
    # Scheduling
    beat_schedule = {
        'rss-ingestion': {
            'task': 'pipeline.tasks.ingest_all_feeds',
            'schedule': crontab(minute=0, hour='*/4'),  # Every 4 hours
        },
        'health-check': {
            'task': 'pipeline.tasks.health_check',
            'schedule': crontab(minute='*/5'),  # Every 5 minutes
        },
        'cleanup-old-jobs': {
            'task': 'pipeline.tasks.cleanup_old_jobs',
            'schedule': crontab(minute=0, hour=2),  # Daily at 2 AM
        }
    }

    # Worker Configuration
    task_serializer = 'json'
    accept_content = ['json']
    result_serializer = 'json'
    timezone = 'UTC'
    enable_utc = True

    # Task Settings
    task_acks_late = True
    worker_prefetch_multiplier = 1
    task_max_retries = 3
    task_default_retry_delay = 60
```

4.4 Deduplication Algorithm

python

```
class DeduplicationEngine:
    def is_duplicate(self, article: Article) -> bool:
        # 1. Exact URL match (after normalization)
        if self.url_exists(self.normalize_url(article.url)):
            return True

        # 2. GUID match
        if self.guid_exists(article.guid):
            return True

        # 3. Content similarity (title + content hash)
        title_hash = hashlib.md5(article.title.lower().strip().encode()).hexdigest()
        content_hash = hashlib.md5(article.body[:1000].encode()).hexdigest()

        if self.content_hash_exists(title_hash, content_hash):
            return True

        # 4. Fuzzy title matching (same publisher, similar title, within 24h)
        if self.fuzzy_title_match(article):
            return True

        return False

    def normalize_url(self, url: str) -> str:
        # Remove tracking parameters, fragments, normalize domain
        parsed = urlparse(url)
        # Remove common tracking parameters
        query_params = parse_qs(parsed.query)
        tracking_params = ['utm_source', 'utm_medium', 'utm_campaign', 'fbclid', 'gclid']
        for param in tracking_params:
            query_params.pop(param, None)

        normalized_query = urlencode(query_params, doseq=True)
        return urlunparse((
            parsed.scheme, parsed.netloc, parsed.path,
            parsed.params, normalized_query, ''
        ))
```

---

5. Quality Standards

5.1 Reliability

- **Uptime**: 99.5% system availability
- **Data Integrity**: 100% - no corrupted or lost articles

- **Duplicate Prevention**: 99.9% accuracy
- **Error Recovery**: Automatic retry with exponential backoff

5.2 Performance

- **Ingestion Speed**: Complete 4-hour cycle within 10 minutes
- **Database Performance**: Query response <100ms
- **Memory Usage**: <512MB per worker process
- **Storage Efficiency**: Minimal redundant data storage

5.3 Monitoring

- **Job Success Rate**: >95% successful ingestion cycles
- **Error Alerting**: Real-time notifications for system failures
- **Performance Tracking**: Historical metrics for optimization
- **Feed Health**: Monitor individual feed performance

---

6. Security & Compliance

6.1 Data Protection

- Encrypted database connections (TLS)
- Secure API key management for external services
- Input validation and sanitization
- SQL injection prevention

6.2 Content Standards

- Respect robots.txt and feed terms of service
- Rate limiting to avoid overwhelming source servers
- Content attribution and source tracking
- Copyright-compliant content handling

---

7. Success Metrics

7.1 Operational Metrics

- **System Uptime**: >99.5%
- **Ingestion Success Rate**: >95%
- **Duplicate Detection Accuracy**: >99.5%
- **Processing Speed**: <10 minutes per 4-hour cycle

7.2 Data Quality Metrics

- **Content Completeness**: >90% articles with all required fields
- **Data Accuracy**: Manual validation of 1% sample shows >99% accuracy
- **Storage Efficiency**: <1% storage waste from duplicates or corrupted data

---

8. Implementation Roadmap

Week 1: Core Infrastructure

- Database schema setup on Neon
- Basic Celery + Redis configuration
- Railway deployment pipeline
- Initial RSS feed parsing logic

Week 2: Deduplication & Quality

- Implement deduplication algorithms
- Add comprehensive error handling
- Database indexing optimization
- Basic monitoring and logging

Week 3: Testing & Hardening

- Load testing with multiple feeds
- Error scenario testing
- Performance optimization
- Documentation and deployment procedures

---

9. Risk Assessment

Technical Risks

- **Feed Reliability**: Mitigation through multiple feed sources and error handling
- **Database Performance**: Mitigation through proper indexing and query optimization
- **Memory Leaks**: Mitigation through worker process recycling and monitoring

Operational Risks

- **Data Loss**: Mitigation through automated backups and transaction safety
- **Cost Overruns**: Mitigation through resource monitoring and auto-scaling limits

- **Feed Access Changes**: Mitigation through feed health monitoring and fallback sources

---

10. Dependencies & Integration Points

External Dependencies

- Neon PostgreSQL (database hosting)
- Railway (deployment platform)
- RSS feed sources (external content providers)

Internal Integration Points

- **Output Interface**: Clean article data available for downstream analysis
- **Monitoring Interface**: Metrics and logs available for dashboard consumption
- **Configuration Interface**: Feed management and system configuration

---

Document Version: 1.0 Last Updated: August 10, 2025 Status: Ready for Implementation