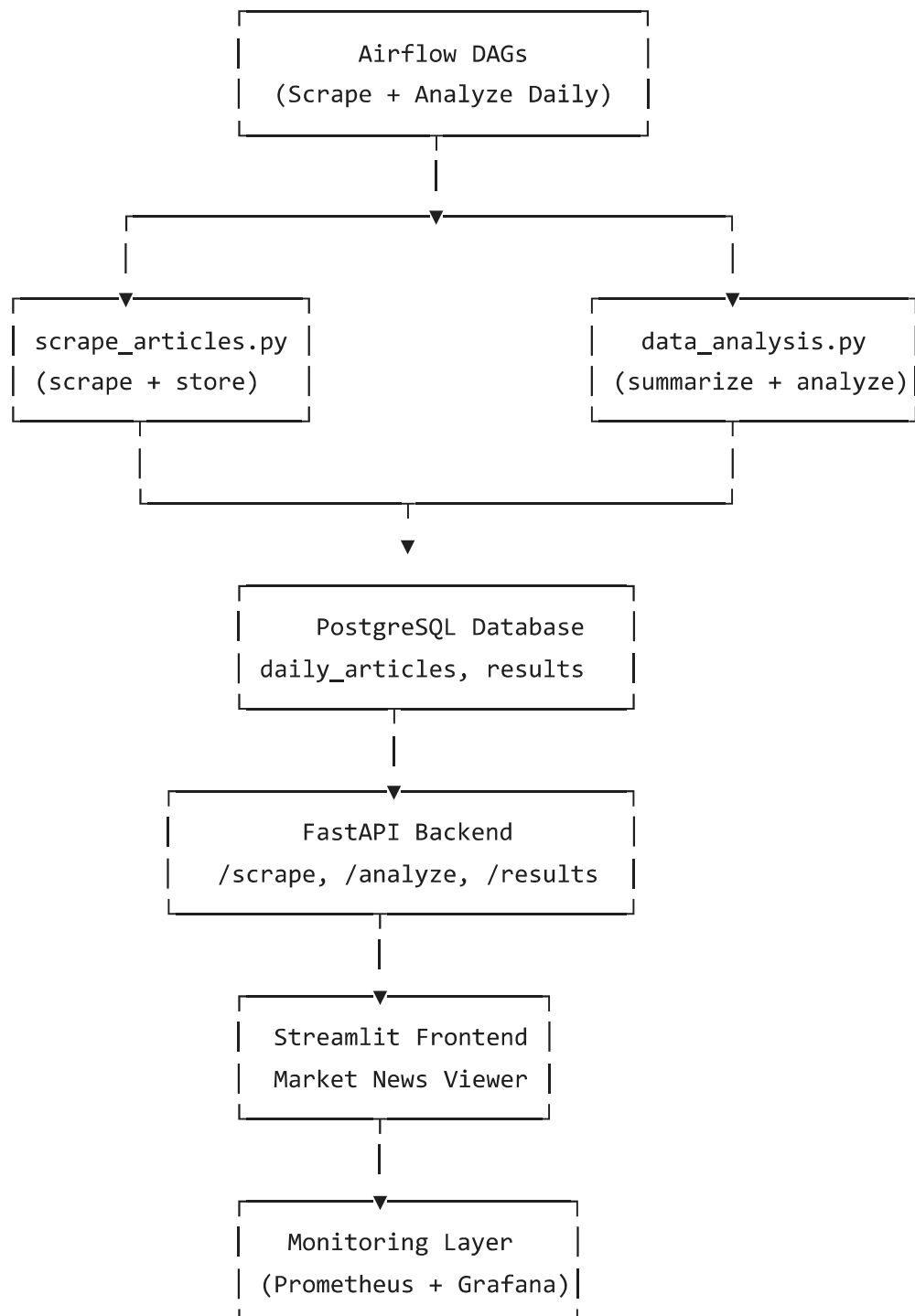


AI-Powered Market News Analysis and Visualization System

Architecture Diagram



1. Component Descriptions

1.1 Airflow DAGs

- Orchestrates and schedules the entire data pipeline.
- Two primary DAGs:
 - `scraper_articles_dag`: Triggers `scrape_articles.py` every day at a fixed time.
 - `analyze_articles_dag`: Triggers `data_analysis.py` post-ingestion.

1.2 `scrape_articles.py`

- Scrapes the top 10 articles from the MoneyControl Markets section.
- Extracts the title, link, and full article content using BeautifulSoup.
- Cleans and stores this data into the `daily_articles` table in PostgreSQL.

1.3 `data_analysis.py`

- Fetches the day's articles from the database.
- Concatenates all article contents and summarizes using `facebook/bart-large-cnn`.
- Applies VADER sentiment analysis.
- Predicts market trend (Positive or Negative).
- Results are stored in the `results` table.

1.4 PostgreSQL Database

- `daily_articles`: Stores scraped data (title, link, content).
- `results`: Stores summarized content, sentiment score, and predicted trend.

1.5 FastAPI Backend

- REST API endpoints:
 - `POST /scrape`: Triggers scraping manually.
 - `POST /analyze`: Triggers analysis manually.
 - `GET /results/{date}`: Fetches results for a specific day.

1.6 Streamlit Frontend

- Visualizes:
 - Title of the day
 - Article summary
 - Sentiment score
 - Market trend

- Targets non-technical users.

2. Monitoring Layer

2.1 Prometheus

- Exposes metrics from:
 - FastAPI server (API call counts, latency)
 - System-level stats (CPU, memory, disk I/O, network usage)
 - PostgreSQL exporter (query performance, connection pool)

2.2 Grafana

- Visual dashboards that monitor:
 - API health (uptime, usage patterns)
 - Resource usage (container stats, memory/cpu)
 - Pipeline run stats (execution success/failures)
 - Ingestion and analysis throughput

2.3 Instrumentation

- FastAPI includes Prometheus middleware to export request/response metrics.
- Exporters used:
 - `prometheus_client` for custom metrics
 - `node_exporter` for server metrics
 - `postgres_exporter` for database performance

3. Key Architectural Benefits

Feature	Benefit
Modularity	Components are loosely coupled and reusable
Observability	Full monitoring stack via Prometheus and Grafana
Automation	Airflow ensures fault-tolerant scheduling of jobs
Flexibility	FastAPI enables manual overrides or ad hoc executions
Accessibility	Non-technical users can use the system via a Streamlit UI
Maintainability	Clear boundaries between scrape, analyze, API, and frontend components

High-Level Design

1. System Overview

The system consists of five primary components:

- 1. Data Ingestion (Scraping)
- 2. Data Analysis (Summarization + Sentiment)
- 3. Storage (PostgreSQL)
- 4. Service Layer (FastAPI)
- 5. Presentation Layer (Streamlit)
- 6. Monitoring (Prometheus + Grafana)
- 7. Orchestration (Airflow)

These components are loosely coupled, allowing for scalability, independent testing, and deployment.

2. Design Choices and Rationale

2.1 Technology Stack

Component	Technology	Rationale
Scraping	Python + BeautifulSoup	Simple, efficient, and well-suited for HTML parsing
Summarization	facebook/bart-large-cnn (Transformers)	State-of-the-art NLP summarization model
Sentiment	VADER (NLTK)	Lightweight, interpretable sentiment model ideal for financial t
Storage	PostgreSQL	Structured relational storage; reliable and ACID compliant
API Layer	FastAPI	High-performance, async-ready, OpenAPI documentation out-c
Frontend	Streamlit	Quick deployment of professional dashboards with minimal fr
Monitoring	Prometheus + Grafana	Production-grade observability stack
Scheduling	Apache Airflow	Industry-standard tool for orchestrating ETL and ML workflow

2.2 Data Model Design

daily_articles Table

- Stores raw scraped data for each date as a JSON blob.
- Fields: date, articles (with title, link, content)

results Table

- Stores processed insights from analysis.
- Fields: date, title, summary, vader_sentiment, market_trend

Rationale:

- Using a two-table system decouples raw data from processed insights, enabling reprocessing and traceability.
-

2.3 Orchestration Strategy

- Apache Airflow schedules:
 - `scrape_articles.py` every morning
 - `data_analysis.py` after scraping
- DAGs ensure reliability, retries, and logs for long-running jobs.

Rationale:

Airflow enables modular, observable pipelines with dependency control, essential for production ETL/ML workflows.

2.4 API Design

- **/scrape**: Manually trigger scraping
- **/analyze**: Manually trigger analysis
- **/results/{date}**: Fetch analysis for a specific day

Rationale:

Separation of scraping and analysis allows independent triggering, monitoring, and debugging. RESTful interface ensures compatibility with external tools and frontends.

2.5 Frontend Design

- Uses Streamlit to display:
 - Title of the day
 - Summary of market articles
 - Sentiment score
 - Market trend indicator

Rationale:

Streamlit allows rapid development of a clean, interactive dashboard ideal for non-technical users without the complexity of building a separate frontend application.

2.6 Monitoring & Observability

- Prometheus scrapes metrics from:
 - FastAPI
 - PostgreSQL exporter

- Node exporter (system metrics)
- Grafana dashboards provide:
 - API uptime
 - Pipeline success/failure rates
 - DB performance
 - System resource utilization

Rationale:

Monitoring ensures production-readiness by detecting failures early, observing usage patterns, and providing performance insights.

2.7 Deployment Strategy

- Modular scripts (`scrape_articles.py` , `data_analysis.py`)
- Dockerized services (optional extension)
- API and UI can be served on different ports/containers
- Scheduler, DB, monitoring components run independently

Rationale:

Allows scaling and managing each component separately. Enables CI/CD, container-based deployment, and cloud migration.

Low-Level Design

1. Base URL

`http://localhost:8000/`

2. API Endpoints Specification

◆ POST `/scrape/`

- **Description:**

Triggers scraping of the top 10 articles from the MoneyControl Markets page and stores them in the `daily_articles` table in PostgreSQL.

- **Request:**

No body parameters.

- **Response:**

```
{
  "message": "✅ Scraped and stored top 10 articles for 2025-04-30"
}
```

- **Error (500):**

```
{
  "detail": "Scraper failed: [error message]"
}
```

- **Database Action:**

Inserts a new record in `daily_articles`:

```
INSERT INTO daily_articles (date, articles) VALUES (<today>, <json_array>);
```

◆ POST /analyze/

- **Description:**

Fetches the day's articles from the `daily_articles` table, summarizes and analyzes them, then stores the result in the `results` table.

- **Request:**

No body parameters.

- **Response:**

```
{
  "message": "✅ Analysis completed and results saved to DB."
}
```

- **Error (500):**

```
{
  "detail": "Analysis failed: [error message]"
}
```

- **Database Action:**

UPSERT into results:

```
INSERT INTO results (date, title, summary, vader_sentiment, market_trend)
VALUES (...)
ON CONFLICT (date) DO UPDATE ...
```

- ◆ GET /results/{date}

- **Description:**


Fetches the summary, sentiment score, and trend for a specific date.

- **Path Parameter:**

- date (required): A string in YYYY-MM-DD format

- **Response:**

```
{
  "date": "2025-04-30",
  "title": "Market Recap: Tech and Energy Sectors Lead Gains",
  "summary": "Markets saw strong gains today with major contributions from technology",
  "vader_sentiment": 0.42,
  "market_trend": "Positive"
}
```



- **Error (404):**

```
{
  "detail": "No result found for date 2025-04-29"
}
```

- **Error (500):**

```
{
  "detail": "Database error: [error message]"
}
```

3. Data Models

◆ daily_articles Table

Column	Type	Description
date	DATE	Date of scraping (PK)
articles	JSON	Array of articles (title, link, content)

Example articles JSON:

```
[
  {
    "title": "Sensex Jumps 500 Points",
    "link": "https://moneycontrol.com/...",
    "content": "Indian equities surged led by gains in IT and banking sectors..."
  },
  ...
]
```

◆ results Table

Column	Type	Description
date	DATE	Primary key
title	TEXT	Summary title for the day
summary	TEXT	Summarized article content
vader_sentiment	FLOAT	Compound sentiment score (-1 to +1)
market_trend	TEXT	Positive or Negative

4. Dependencies

- psycopg2-binary – for PostgreSQL DB connection
- fastapi – API framework
- transformers – Summarization model
- vaderSentiment – Sentiment analysis
- uvicorn – Server runner

5. Validation and Error Handling

- All endpoints handle exceptions and return appropriate HTTP status codes.
- Input date formats are strictly validated (YYYY-MM-DD).
- Database connection and query errors are wrapped in try/except blocks.

✓ Test Plan & Test Cases

1. Test Scope

- FastAPI endpoints (/scrape , /analyze , /results)
- Streamlit frontend rendering
- Database integrity (PostgreSQL)
- Airflow DAG trigger (optional manual test)
- Monitoring availability (Prometheus endpoint)

2. Test Environment

- **OS:** Windows/Linux/macOS
- **Python Version:** 3.9+
- **PostgreSQL:** v12+
- **FastAPI:** v0.95+
- **Streamlit:** v1.20+
- **Monitoring:** Prometheus + Grafana
- **Tooling:** cURL, Postman, Web browser

3. Test Cases

FastAPI API Tests

ID	Test Case	Endpoint	Expected Output	Result
TC01	Trigger scrape manually	POST /scrape/	200, message with today's date	✓/✗
TC02	Trigger analysis manually	POST /analyze/	200, success message	✓/✗
TC03	Query today's result	GET /results/{date}	200, JSON with title, summary, sentiment, trend	✓/✗
TC04	Query invalid date	GET /results/1999-01-01	404 not found	✓/✗
TC05	API fails if DB is down	All	500 error response	✓/✗

Streamlit Frontend Tests

ID	Test Case	Action	Expected Output	Result
TC06	Load Streamlit dashboard	Open /streamlit_app.py	Page loads with title, summary, indicators	✓/✗
TC07	Show warning when no data exists	No data in DB	"No analysis available" warning	✓/✗
TC08	Renders summary and trend correctly	Summary present	Proper formatting, styling, scores visible	✓/✗

Database Tests

ID	Test Case	Triggered By	Expected Output	Result
TC09	Insert into <code>daily_articles</code>	<code>/scrape/</code>	Row with 10 articles in JSON	✓/✗
TC10	Insert into <code>results</code>	<code>/analyze/</code>	Row with correct summary/sentiment	✓/✗
TC11	Prevent duplicate insert on same day	Repeat calls	UPSERT behavior (row updates not fails)	✓/✗

Monitoring Tests

ID	Test Case	Target	Expected Output	Result
TC12	Check FastAPI Prometheus metrics	<code>/metrics</code>	Exposes request count, latency, etc.	✓/✗
TC13	Check Grafana dashboards	Grafana UI	Show API uptime, memory usage, etc.	✓/✗

5. Test Automation Suggestions

- Use `pytest` + `requests` for automated API tests
- Use `unittest` + `mock` for backend scripts
- Use `watchdog` + shell scripts to test DAG triggers

User Manual (Non-Technical User)

Welcome to the Market News Summary Dashboard

This application summarizes the top 10 financial articles every day from MoneyControl and tells you whether the market is likely to trend positively or negatively – all in simple language.

What You Can Do

View Today's Market Summary

1. Open the website:
<http://localhost:8501>
2. You will see:
 - **Title of the day** (e.g., "Markets Rally After RBI Decision")
 - **Summary:** A few paragraphs summarizing all 10 articles.
 - **Market Trend:** "Positive" or "Negative"
 - **Sentiment Score:** From -1 (very negative) to +1 (very positive)

How It Works Behind the Scenes

- The system **automatically fetches news every day**
 - It **analyzes and summarizes the news**
 - You get a **simple report**, generated by AI
-

If Something Looks Outdated

- Ask the admin to:
 - Go to: <http://localhost:8000/docs>
 - Run:
 - POST /scrape → to get the latest articles
 - POST /analyze → to generate today's summary
-

Troubleshooting

Issue	What to Do
Page says "No analysis available"	Ask admin to rerun the pipeline
Dashboard does not load	Check if the Streamlit server is running
Numbers look wrong	Wait for Airflow or re-analyze manually

Double-click (or enter) to edit