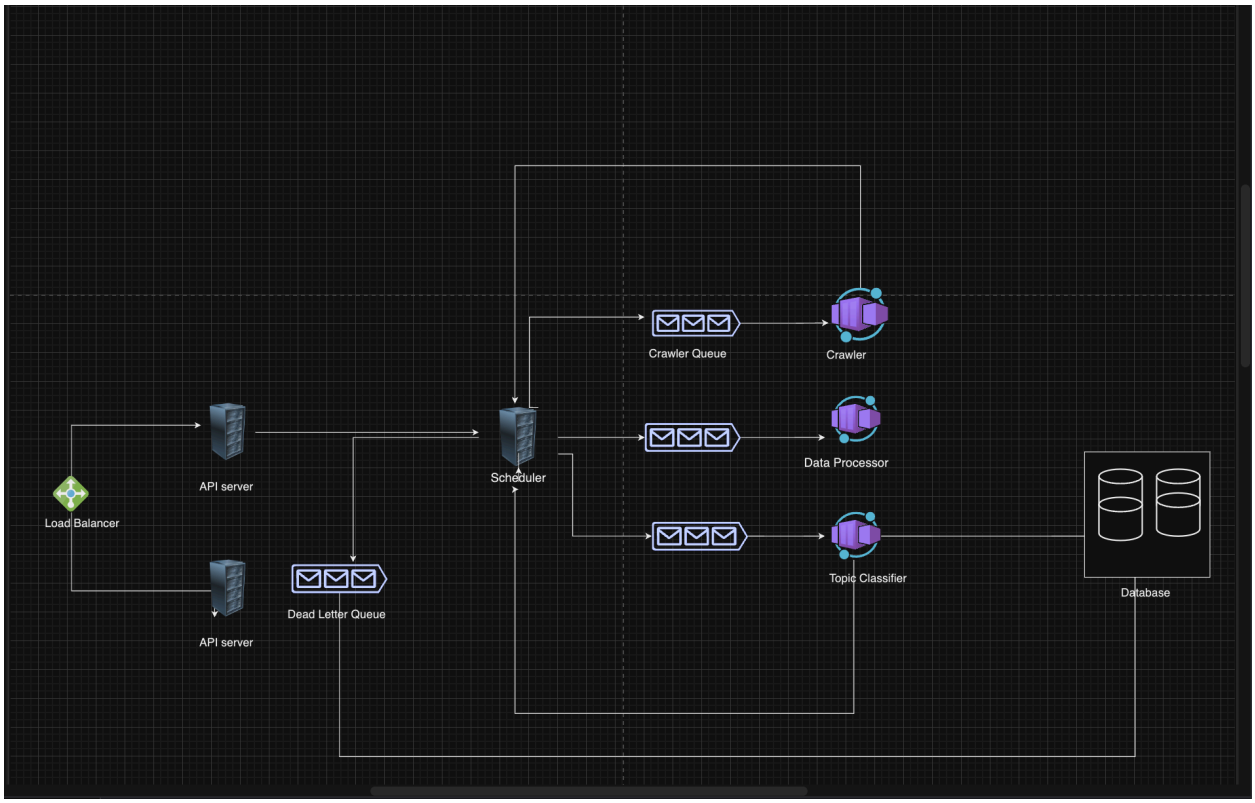
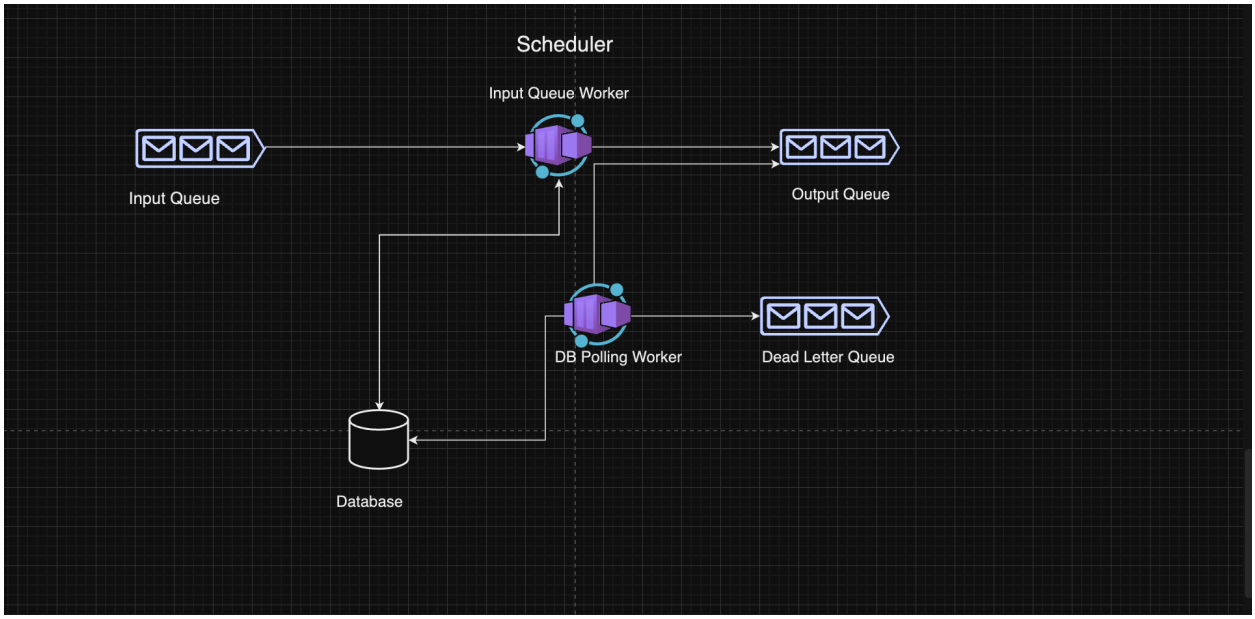
Crawler & Keyword Extraction System





Introduction

This document outlines the design for a scalable and resilient system for processing submitted links. The system comprises an API server, a scheduler service, a crawler service, a data processing service, and a keyword extractor service. Kafka is utilized as the messaging queue for inter-service communication to ensure reliability and scalability.

System Components

1. API Server

The API server exposes a `submit-link` route that accepts POST requests with the following body:

```
{
  "link": "www.abc.com",
  "api-dev-key": "your-token"
}
```

To enhance availability, multiple instances of the API server can be deployed behind a load balancer.

Responsibilities:

- Accept incoming link submissions.
- Validate the request and ensure the presence of a valid API development key.
- Write the validated message to the Scheduler Service input queue.

2. Scheduler Service

The Scheduler Service manages the flow of messages between services and ensures retry logic for failed operations. It leverages **Kafka** for message queuing.

Components:

- **Input Queue**: Listened to by input queue workers. The number of partitions can be adjusted to scale processing.
- **Input Queue Workers**: Responsible for writing messages to the appropriate downstream service queues and scheduling retries on failure.
- **Database**: Stores messages scheduled for retry.
- **Polling Worker**: Periodically checks the database for retry-scheduled messages and redirects them as necessary.
- **Dead Letter Queue**: Receives messages that have exceeded the retry threshold for further analysis.

Retry Strategy:

- **Exponential Backoff**: Retries for the Crawler Service follow an exponential backoff strategy to reduce load and collision.

3. Crawler Service

The Crawler Service fetches and processes HTML and text content from the submitted links.

Responsibilities:

- **Fetching HTML Content**: Retrieve the HTML content of the given URL.
- **Parsing and Extracting Text**: Extract meaningful text content from the HTML.
- **Handling Dynamic Content**: Handle dynamically generated content.
- **Customization for Specific Sites**: Optionally, read configurations for site-specific parsing rules.
- **Rate Limiting**: Implement rate limiting and respect `robots.txt` to avoid IP bans.
- **Error Handling**: Gracefully handle HTTP errors, timeouts, and retries.
- **User-Agent Spoofing**: Use different User-Agent strings to mimic different browsers and avoid detection.

4. Data Processing Service

The Data Processing Service cleans the extracted text content.

Responsibilities:

- **Text Cleanup**: Remove stop words, ads, and other extraneous content from the text.
- **Handling Busy Webpages**: Implement strategies to clean up content from pages with a high density of interactive elements and distractions.

Strategies for Cleaning Up Content from Busy Webpages

1. HTML Parsing and Tag Filtering
2. Text Extraction Libraries: Use libraries like Readability, Goose, or Newspaper3k that specialize in extracting main content from webpages.
3. Regular Expressions and Pattern Matching: Use regular expressions to identify and remove common patterns associated with unwanted content, such as navigation links and advertisements.
4. Natural Language Processing (NLP) Techniques: Apply named entity recognition (NER) and part-of-speech (POS) tagging to differentiate between meaningful content and fluff.

5. Keyword Extractor Service

The Keyword Extractor Service classifies the cleaned text content and extracts keywords.

Techniques:

- **TF-IDF (Term Frequency-Inverse Document Frequency)**: Measures the importance of words in a document relative to a corpus.
- **LDA (Latent Dirichlet Allocation)**: A generative statistical model for topic discovery.
- **LLM (Large Language Models)**: Advanced models for understanding and generating human language.

Kafka as the Messaging Queue

Kafka is chosen for its robust messaging capabilities, offering the following benefits:

- **Scalability**: Kafka can handle large volumes of data with low latency.
- **Durability**: Messages are replicated across brokers, ensuring high availability.
- **Fault Tolerance**: Kafka's partitioning and replication mechanisms provide resilience against failures.
- **High Throughput**: Suitable for real-time data processing.

Design Considerations

Loose Coupling

The system's microservices architecture ensures loose coupling, allowing individual services to scale independently and evolve without impacting others.

Scalability

- **Input Queue Partitions**: The number of partitions in Kafka queues can be adjusted to scale worker instances.
- **Service Replication**: Services can be replicated across multiple data centers or regions for enhanced availability.

Failure Handling

- **Retries and Exponential Backoff**: Retries with exponential backoff reduce the risk of overwhelming services and ensure efficient resource use.
- **Dead Letter Queue**: Captures unprocessable messages for further analysis, ensuring that issues can be addressed without disrupting the system.

Deployment and Replication Strategies

- **Intra-Data Center Replication**: Ensures high availability within a single data center.
- **Inter-Data Center Replication**: Provides resilience across multiple data centers.
- **Multi-Region Replication**: Enhances global availability and disaster recovery capabilities.

This design ensures a robust, scalable, and fault-tolerant system for processing submitted links. By leveraging Kafka for messaging and implementing microservices with clear responsibilities, the system achieves high availability and efficient processing.