High-Level Design Document: Typeahead Application

1. Introduction

The Typeahead Application provides real-time suggestions as users type. It leverages a microservices architecture to handle continuous input processing, data aggregation, and real-time querying, ensuring scalability and efficiency.

Purpose

This document outlines the high-level architecture, components, and design principles of the Typeahead Application. It serves as a reference for understanding the system's structure, functionality, and technology stack.

Scope

The document covers:

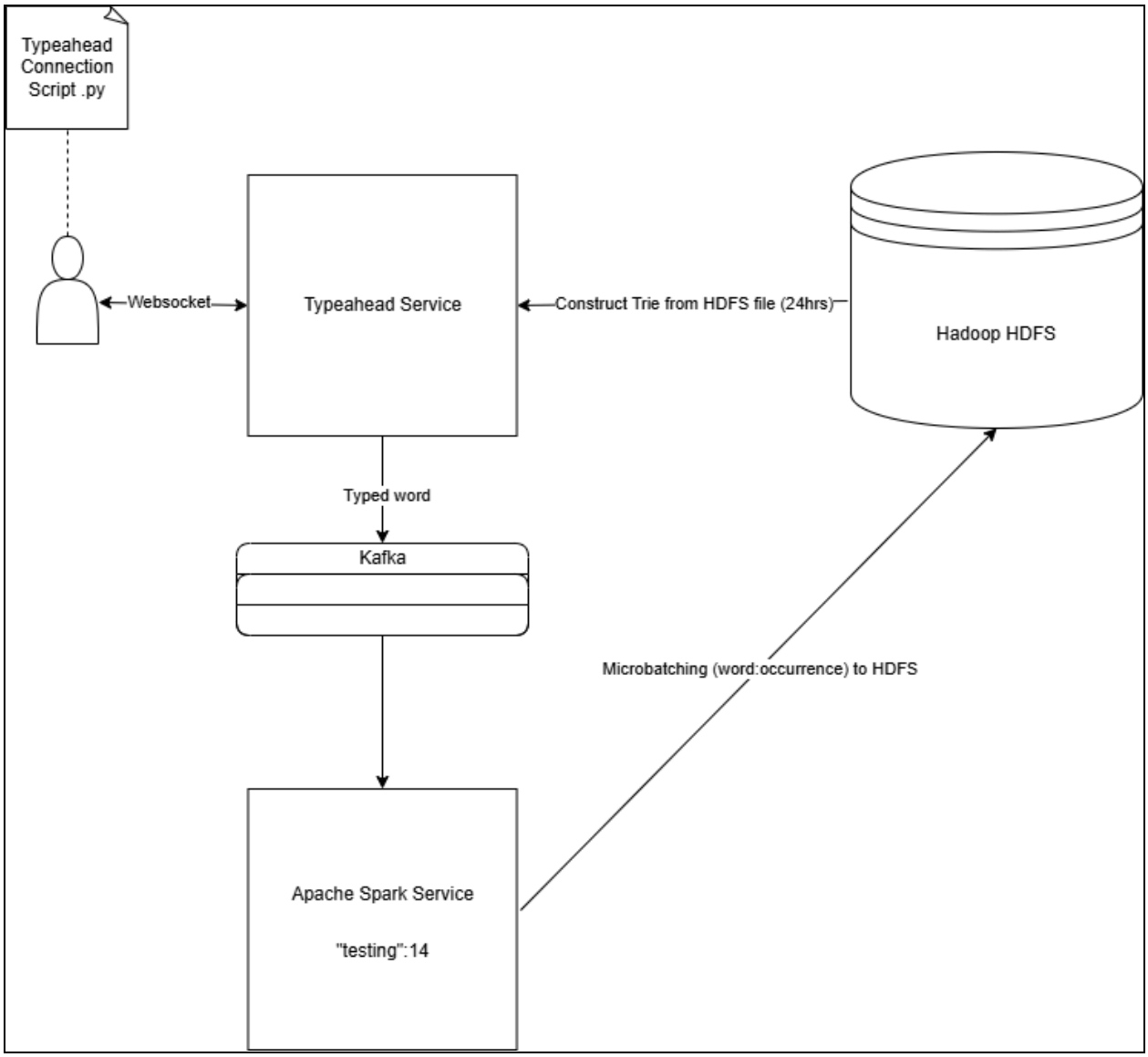
- System architecture and components
- Data flow and interaction between services
- Technologies used and deployment strategies

2. System Overview

The Typeahead Application is designed to provide quick, real-time suggestions by utilizing a Trie data structure. It uses distributed processing and scalable architecture, built using Spring Boot, Kafka, Spark, HDFS, and Docker.

3. Architecture Design

3.1 Architectural Diagram



3.2 Components

1. **Typeahead Service**
    - **Technology**: Spring Boot, Java
    - **Description**: Integrates a Trie data structure for fast prefix lookup. It provides a WebSocket endpoint to receive user input and offer suggestions in real-time.
    - **Key Features**:
        - Handles WebSocket connections for low-latency communication.
        - Sends typed prefixes to the Kafka messaging queue for processing.
2. **Apache Kafka**
    - **Technology**: Apache Kafka
    - **Description**: Acts as a messaging queue between the Typeahead Service and Spark Streaming Service.

    - **Key Features**:
        - Reliable delivery of typed prefixes.
        - Scales to handle multiple streams from different sources.
3. **Spark Streaming Service**
    - **Technology**: Apache Spark
    - **Description**: Consumes prefixes from Kafka and processes them in micro-batches. It aggregates the count of each prefix and updates the HDFS with popular prefixes.
    - **Key Features**:
        - Efficient batch processing of stream data.
        - Updates HDFS to store the aggregated counts of popular prefixes.
4. **Hadoop HDFS**
    - **Technology**: Apache Hadoop
    - **Description**: Distributed file storage system for storing aggregated prefix data. Spark writes the most popular prefixes here, which are loaded by the Typeahead Service for daily updates.
5. **Docker**
    - **Technology**: Docker
    - **Description**: Containerization platform to deploy services. Enables easier scaling and management of multiple instances of the Typeahead and Spark Streaming services.
6. **Python Script (Client)**
    - **Technology**: Python
    - **Description**: CLI script that establishes a WebSocket connection to interact with the Typeahead service and receive real-time suggestions.

4. Data Flow

4.1 User Interaction

1. Users type characters via a client (CLI).
2. The input is sent to the Typeahead Service through a WebSocket connection.
3. The Typeahead Service queries the Trie data structure for matching prefixes and returns suggestions.

4.2 Prefix Processing

1. The Typeahead Service sends aggregated prefixes (words) to Kafka.
2. The Spark Streaming Service consumes these words, processes them in micro-batches, and aggregates the counts.
3. Processed data is then written to HDFS.

4.3 Trie Data Structure Updates

1. Every 24 hours, the Typeahead Service reads the updated word data from HDFS.
2. The Trie is rebuilt with the latest popular words to ensure accurate suggestions.

5. Scalability and Fault Tolerance

- **Scalability**: Dockerized deployment allows for horizontal scaling of services. Kafka ensures scalable message processing, while Spark handles distributed stream processing.
- **Fault Tolerance**: Kafka's reliable delivery ensures messages are processed without loss. Spark can recover from processing failures by restarting from the last checkpoint.

6. Technologies Used

- **Spring/Spring Boot**: Backend framework for REST APIs and WebSocket support.
- **Apache Kafka**: Reliable message broker.
- **Apache Spark**: Stream processing for aggregating prefix data.
- **Trie Data Structure**: Efficient searching for prefix suggestions.
- **Hadoop HDFS**: Storage system for persistent prefix data.
- **Docker**: Containerization for easy deployment and scaling.
- **JUnit/Mockito**: Testing framework for robust unit and integration tests.
- **Python**: Client-side script for establishing WebSocket connection via CLI.

7. Deployment Strategy

- **Docker Containers**: Each component runs in a separate container, enabling easy deployment, scaling, and management.
- **Kubernetes (Optional)**: Consider using Kubernetes for orchestrating Docker containers if scaling becomes a requirement.

8. Conclusion

The Typeahead Application's architecture enables efficient real-time processing and scalable suggestions. By decoupling components using Kafka and deploying services as Docker containers, it provides flexibility for scaling and maintenance.