**Step-by-Step Guide to Setting up a Data Pipeline**

**Table of Contents**

1. Introduction

2. Architecture

3. Environment Setup

4. Service Deployments in Minikube

5. Configuring Kafka Connect and Debezium for CDC

**1. Introduction**

**Project Overview:**

This project aims to build a comprehensive end-to-end data pipeline that leverages various data engineering tools to handle real-time data processing. The primary goal is to extract data from a MySQL database using Change Data Capture (CDC) techniques with Debezium, transport this data using Kafka, process it using Apache Spark, and store the final output in a file-based storage system. The project demonstrates a robust approach to ingesting and processing data in real-time, ensuring that database changes are immediately reflected in downstream systems for further processing and analysis.

**Goals and Objectives:**

* Real-time Data Ingestion: Capture changes in the MySQL database using Debezium for CDC and stream them into Kafka.
* Scalable Processing: Use Apache Spark to process the real-time data streams efficiently.
* Data Persistence: Store the processed data as JSON files and dump them into a scalable storage solution (e.g., HDFS or S3).
* Fault Tolerance and Scalability: Ensure the pipeline is resilient to failures and can scale horizontally as data volume increases.
* Tools and Technologies:
* MySQL: The source database, where transactional data resides.
* Debezium: A distributed platform for CDC, capturing row-level changes in MySQL and pushing them to Kafka.
* Kafka: A distributed messaging system that captures and streams changes in real-time from MySQL.
* Apache Spark: A big data processing engine that reads Kafka streams, processes JSON messages, and writes the output to storage.
* Storage: A file-based storage system such as HDFS or Amazon S3, where processed data is stored for downstream analytics.
* Kubernetes (Minikube): The local environment where all components are orchestrated.

**2. Architecture**

System Architecture Diagram:

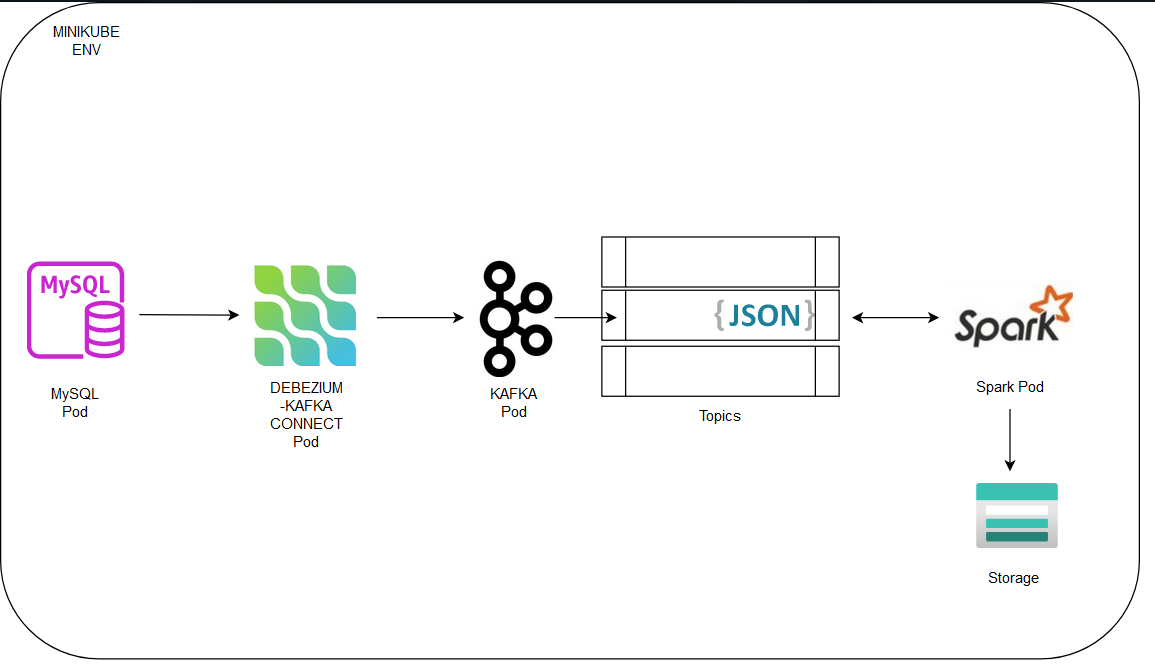
* Include a diagram here depicting the data flow between MySQL, Debezium, Kafka, Spark, and the storage system.

Component Overview:

* MySQL: A relational database management system that stores transactional data. The data changes (INSERT, UPDATE, DELETE) in MySQL are the main trigger for the pipeline.
* Debezium: A CDC tool that monitors the MySQL binlog and captures any change events. These events are then published as messages into Kafka topics.
* Kafka: A high-throughput distributed messaging platform. Kafka is responsible for transporting the captured CDC events from Debezium to Spark. It ensures that data can be processed in real-time or near real-time as it changes in the database.
* Apache Spark: A distributed processing framework. In this project, Spark is used to consume Kafka streams, process the incoming JSON data (representing CDC events), and apply necessary transformations or enrichments. After processing, the data is written into a file-based storage system.
* File Storage: The processed data is stored as JSON files in a storage system such as HDFS or S3. This provides durability and allows the processed data to be used for further analysis or batch processing.

Data Flow:

* Source (MySQL): Changes in the MySQL database (e.g., new records, updates, or deletions) are logged into the binlog, which Debezium monitors.
* CDC with Debezium: Debezium acts as a CDC connector and captures changes in MySQL in real-time. These changes are transformed into JSON events, which Debezium publishes to Kafka topics.
* Streaming with Kafka: Kafka receives the JSON events from Debezium and streams them in real-time. Each topic in Kafka corresponds to a table in MySQL, and any change in MySQL will be published to its respective Kafka topic.
* Processing with Spark: Spark consumes the Kafka streams in real-time, processes the incoming JSON messages (applying any transformations or enrichments), and writes the output to a file system in batches.
* Storage: Finally, the processed JSON files are written to a storage system such as HDFS or S3. This output can then be accessed for analytics, reporting, or further batch processing.

****

**3. Environment Setup**

**Step 1:** Install Docker Desktop

Docker Desktop is essential to run containers locally. Minikube will use Docker as its container runtime.

Download and Install Docker Desktop:

Go to the Docker Desktop download page and download the installer for your operating system.

Follow the installation instructions for your specific platform (Windows, macOS, or Linux).

Start Docker Desktop:

After installation, launch Docker Desktop.

Ensure that Docker is running properly by opening a terminal and typing:

css

Copy code

You should see the version information if Docker is running correctly.

**Step 2:** Install Minikube

Minikube creates a Kubernetes cluster locally for testing and development.

Download and Install Minikube:

Go to the Minikube installation guide and download the appropriate binary for your operating system.

You can use a package manager like brew for macOS or choco for Windows:

macOS (with Homebrew):



Windows (with Chocolatey):



Verify Minikube Installation:

Check if Minikube is installed correctly by running the following command:



**Step 3:** Start Minikube

Once Docker Desktop and Minikube are installed, you can start Minikube using Docker as the container runtime.

Launch Minikube with Docker:

In the terminal, run the following command to start Minikube using Docker as the driver:

Minikube will set up a local Kubernetes cluster that uses Docker as the container engine.

Check Cluster Status:

Verify that Minikube is running correctly by typing:

You should see the status of the Kubernetes cluster (components like kubelet, apiserver, etc.).

**Step 4:** Interact with Kubernetes using kubectl

kubectl is the command-line tool used to manage Kubernetes clusters.

Minikube installs kubectl automatically, but you can also install it separately. To verify, type:

Access Kubernetes Dashboard:

Minikube provides a simple web-based UI for managing the cluster. To launch the Kubernetes dashboard, run:

This will open the Kubernetes dashboard in your web browser.

**4. Service Deployments in Minikube**

**Step 1:** Deploy MySQL in Minikube

Create MySQL Deployment YAML:

First, you need to define a YAML file for deploying MySQL in the Kubernetes cluster. Here's an example mysql-deployment.yaml in folder MySQL

Apply MySQL Deployment:

Use the following command to apply the MySQL deployment:

Verify MySQL Deployment

Check the status of the MySQL pod by running

**Step 2:** Deploy Zookeeper in Minikube

Create Zookeeper Deployment YAML:

Here’s a simple YAML file for Zookeeper (zookeeper-deployment.yaml):

Deploy Zookeeper with:

Verify Zookeeper Deployment:

Check if Zookeeper is running:

**Step 3:** Deploy Kafka in Minikube

Create Kafka Deployment YAML:

Deploy Kafka with:

Verify Kafka Deployment:

Use this command to check if Kafka is running:

**Step 4:** Deploy Spark in Minikube

Create Spark YAML:

Here’s a basic Spark deployment file (spark-deployment.yaml):

Deploy Spark by running:

Verify Spark Deployment

**Step 5:** Connect Services

Access Kafka: To access Kafka topics and test the connectivity:

Access MySQL: You can connect to MySQL for testing:

**Step 6:** Monitor the Deployments

To check the status of all deployed services (MySQL, Zookeeper, Kafka, Spark):

kubectl get all

**5. Configuring Kafka Connect and Debezium for CDC**

Now that we have the services running in Minikube (MySQL, Zookeeper, Kafka, Spark), the next step is to configure Kafka Connect and Debezium to capture Change Data Capture (CDC) events from MySQL and stream them to Kafka.

**Step 1:** Deploy Kafka Connect with Debezium

Create Kafka Connect Deployment YAML:

This deployment includes Kafka Connect with the Debezium connector to capture changes from MySQL:

Apply Kafka Connect Deployment:

Deploy Kafka Connect by applying the YAML file:

Check if Kafka Connect is running

**Step 2:** Configure Debezium MySQL Connector

Prepare MySQL for CDC:

Ensure that MySQL is configured with binlog enabled, which is necessary for Debezium to capture changes. You can check this with:

Enable the binary log if it’s not already enabled by adding the following to the my.cnf file (you might need to restart MySQL):

Register MySQL Debezium Connector:

You need to register the Debezium connector with Kafka Connect to start capturing CDC events. Use the following API call to register the connector:



Make sure you replace your\_password, your\_database, and <kafka-connect-pod-ip> with the actual values.

**Verify Connector Registration:**

You can verify if the connector was registered successfully by checking the status:



After registering the connector, Debezium will begin capturing changes from the MySQL database and will send them as JSON events to Kafka topics.

You can list the Kafka topics and check if the CDC events are being published:

**Step 3:** Test the Data Flow

Make Changes in MySQL:

Insert, update, or delete rows in the MySQL database to generate CDC events.

**Step 4:** Consume CDC Events from Kafka:

You can consume the events that Kafka has captured using the following command:

Enter the Kafka Pods using:

To find the list of topics:

The below command will help you see the incoming messages:



The console should display the JSON messages with the data changes.

**Step 5:** Integrate with Spark

Configure Spark to Consume Kafka Streams:

In your Spark environment, use the Spark-Kafka integration to consume the Kafka streams. A typical code snippet looks like this: