Data Engineering Assignment

Xây dựng 1 nền tảng dữ liệu đơn giản

Viettel Digital Talent 2024

Link source code

Lĩnh vực	Data Engineering
Thực tập sinh	Đào Anh Vũ
Số thứ tự	37
Academic Supervisor	Phạm Công Minh

Trong nền tảng dữ liệu chúng ta đang xây dựng, có 4 thành phần chính đó là: Kafka, Nifi, Hadoop và Spark. Bản báo cáo này sẽ đi qua các bước triển khai, chi tiết từng thành phần cũng như giải thích những service đi kèm của chúng. Tất cả chi tiết của source code đều có thể tìm thấy ở link.

1 Bước 1: Triển khai Kafka và Producer.

1.1 Kafka

```yaml
zookeeper:
  image: confluentinc/cp-zookeeper:latest
  environment:
    ZOOKEEPER_CLIENT_PORT: 2181
    ZOOKEEPER_TICK_TIME: 2000
  ports:
    - 2181:2181
  container_name: zookeeper
  volumes:
    - ./zoo_volume/data:/var/lib/zookeeper/data
    - ./zoo_volume/log:/var/lib/zookeeper/log

broker01:
  image: confluentinc/cp-kafka:latest
  depends_on:
    - zookeeper
  container_name: broker01
  environment:
    KAFKA_BROKER_ID: 1
    KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
    KAFKA_LISTENERS: PLAINTEXT_HOST://broker01:9092,PLAINTEXT://broker01:9093,PLAINTEXT_BROKER://broker01:9094
    KAFKA_ADVERTISED_LISTENERS: PLAINTEXT_HOST://broker01:9092,PLAINTEXT://broker01:9093,PLAINTEXT_BROKER://localhost:9094
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT,PLAINTEXT_BROKER:PLAINTEXT
    KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
    KAFKA_NUM_PARTITIONS: 2
    KAFKA_AUTO_CREATE_TOPICS_ENABLE: "true"
    KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
    KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 1
    KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 1
    KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 100
  ports:
    - 9092:9092
    - 9094:9094
  volumes:
    - ./kafka_volume/data:/var/lib/kafka/data
```

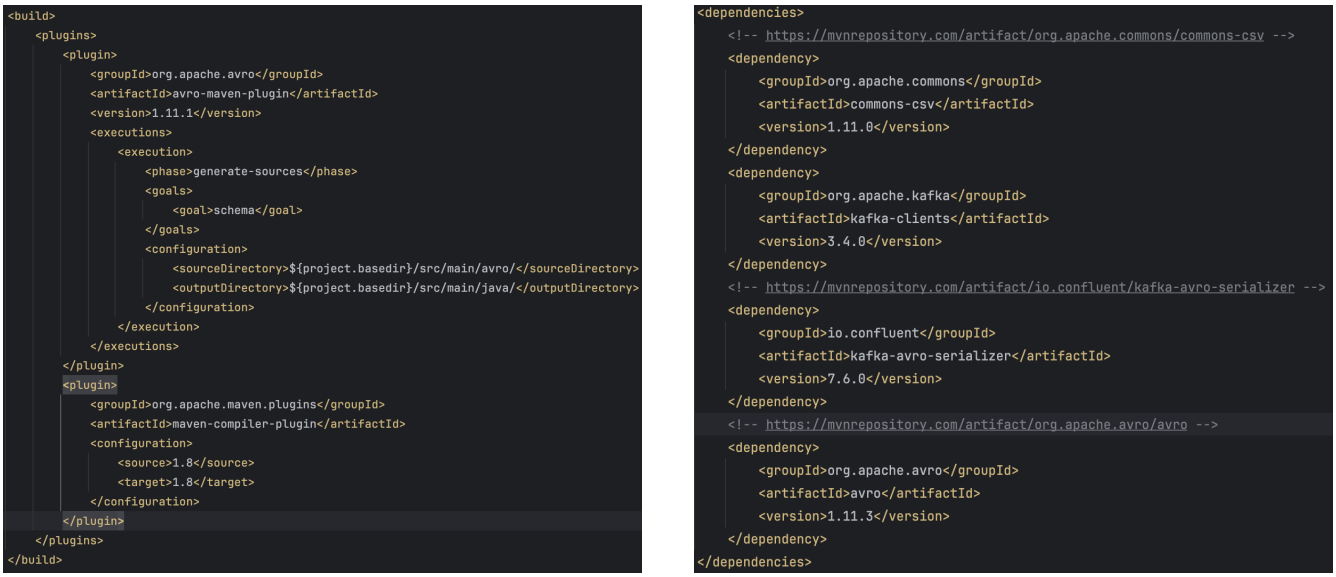
```yaml
kafka-ui:
  container_name: kafka-ui
  image: provectuslabs/kafka-ui:latest
  ports:
    - 8080:8080
  depends_on:
    - broker01
  environment:
    KAFKA_CLUSTERS_0_NAME: my-kafka-ui
    KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS: broker01:9093

my-schema-registry:
  image: confluentinc/cp-schema-registry:latest
  hostname: my-schema-registry
  container_name: my-schema-registry
  ports:
    - "8081:8081"
  environment:
    SCHEMA_REGISTRY_KAFKASTORE_BOOTSTRAP_SERVERS: "broker01:9093"
    SCHEMA_REGISTRY_HOST_NAME: my-schema-registry
    SCHEMA_REGISTRY_LISTENERS: http://localhost:8081,http://my-schema-registry:8081
  depends_on:
    - broker01
```

Hình 1: Các services liên quan đến Kafka.

- Zookeeper và Kafka Broker: Trong dự án này, em đã chọn sử dụng Zookeeper thay cho Kraft vì khi triển khai cả hai, em cảm thấy Kraft không ổn định bằng Zookeeper. Ngoài ra, do quy mô của bài tập không quá lớn, nên em thấy chỉ cần sử dụng một message broker là đủ.
- Kafka-UI: Đây là service được sử dụng để truy cập vào Kafka thông qua một giao diện web hoàn chỉnh. Nó cung cấp đầy đủ các tính năng để tương tác với Kafka.
- Schema Registry: Service này được dùng để đăng ký schema cho Kafka. Nhờ vào nó, cả chương trình Java (Producer) và Nifi (Consumer) đều có thể nhận biết những schema đã được định sẵn. Service này sử dụng Avro để mã hóa và biên dịch dữ liệu dựa trên file `.avsc`, tạo ra tính đồng nhất giữa các chương trình.

Sau khi triển khai những service nêu trên, Kafka đã sẵn sàng được sử dụng ở máy tính cá nhân bằng đường dẫn `localhost:9094`, hoặc qua các container với `broker01:9092`. Ngoài ra, chúng ta có thể tương tác với nó với giao diện web ở `localhost:8080`. Tiếp đến, chúng ta sẽ đi qua một chương trình Java đơn giản dùng để gửi các records vào Kafka.

1.2 Chương trình Java để đẩy mesaage vào Kafka (Producer)



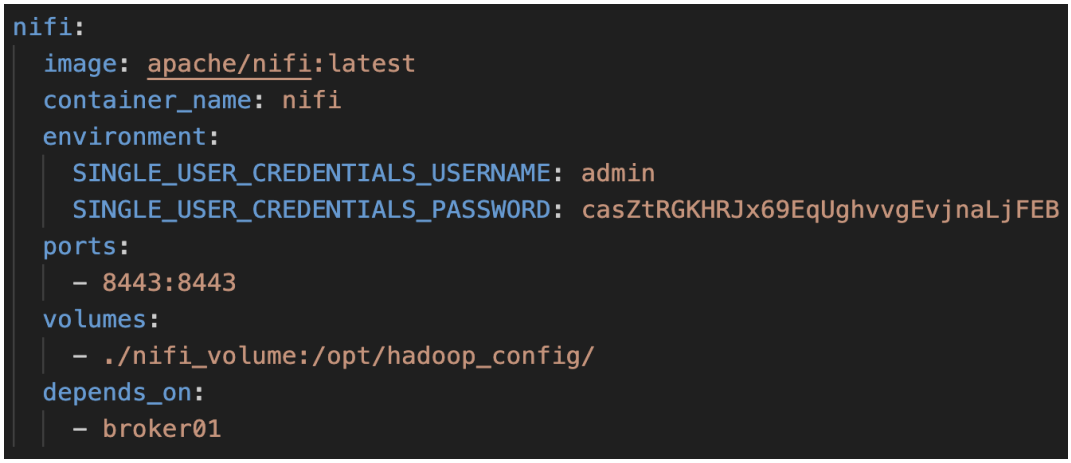
```xml
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.avro</groupId>
            <artifactId>avro-maven-plugin</artifactId>
            <version>1.11.1</version>
            <executions>
                <execution>
                    <phase>generate-sources</phase>
                    <goals>
                        <goal>schema</goal>
                    </goals>
                    <configuration>
                        <sourceDirectory>${project.basedir}/src/main/avro/</sourceDirectory>
                        <outputDirectory>${project.basedir}/src/main/java/</outputDirectory>
                    </configuration>
                </execution>
            </executions>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
                <source>1.8</source>
                <target>1.8</target>
            </configuration>
        </plugin>
    </plugins>
</build>
```

```xml
<dependencies>
    <!-- https://mvnrepository.com/artifact/org.apache.commons/commons-csv -->
    <dependency>
        <groupId>org.apache.commons</groupId>
        <artifactId>commons-csv</artifactId>
        <version>1.11.0</version>
    </dependency>
    <dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-clients</artifactId>
        <version>3.4.0</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/io.confluent/kafka-avro-serializer -->
    <dependency>
        <groupId>io.confluent</groupId>
        <artifactId>kafka-avro-serializer</artifactId>
        <version>7.6.0</version>
    </dependency>
    <!-- https://mvnrepository.com/artifact/org.apache.avro/avro -->
    <dependency>
        <groupId>org.apache.avro</groupId>
        <artifactId>avro</artifactId>
        <version>1.11.3</version>
    </dependency>
</dependencies>
```

Hình 2: Yêu cầu cho file `pom.xml`

Producer có thể được tìm thấy trong thư mục `activity-producer`, một project Maven cơ bản. Để chuẩn bị cho phần này, trước tiên chúng ta cần cài đặt một số plugins để sử dụng Avro cho việc mã hoá. Các plugins này sẽ tự động chuyển file `.avsc` sang code Java. Sau khi hoàn tất chuẩn bị, mỗi khi tạo ra một file Avro mới, chúng ta chỉ cần chạy lệnh `mvn generate-sources` để các class Java được tự động tạo ra và sẵn sàng sử dụng nhờ các builder có sẵn. Về các dependencies, chúng bao gồm những thư viện cơ bản để xử lý file CSV, kết nối với Kafka và làm việc với Avro. Sau khi tải tất cả các dependencies về, chúng ta có thể vào file `ActivityProducer.java` và chạy main method. Chương trình này sẽ đẩy từng record của file `log_action.csv` vào Kafka mỗi giây.

2 Bước 2: Triển khai Nifi.



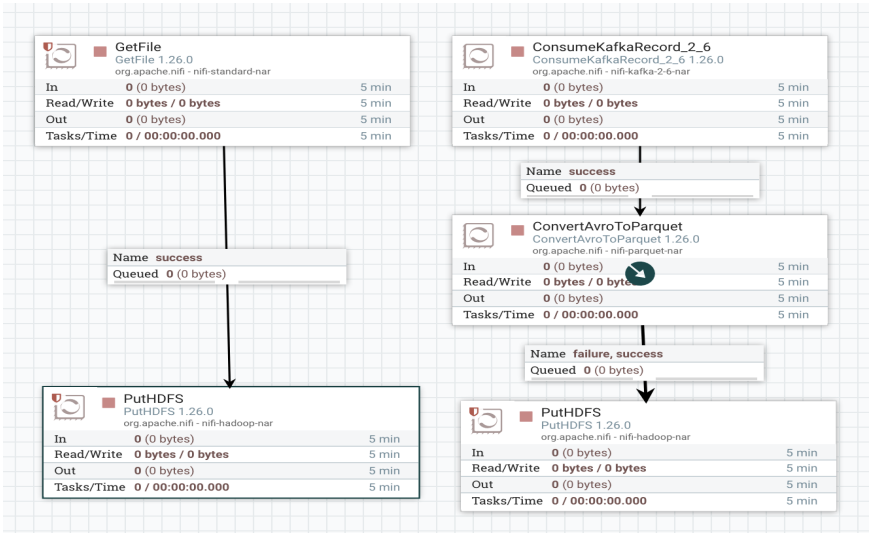
```yaml
nifi:
  image: apache/nifi:latest
  container_name: nifi
  environment:
    SINGLE_USER_CREDENTIALS_USERNAME: admin
    SINGLE_USER_CREDENTIALS_PASSWORD: casZtRGKHRJx69EqUghvvgEvjnaLjFEB
  ports:
    - 8443:8443
  volumes:
    - ./nifi_volume:/opt/hadoop_config/
  depends_on:
    - broker01
```

Hình 3: Cấu hình của Nifi trong docker-compose.

Chúng ta sẽ sử dụng Nifi để chuyển các record từ Kafka sang dạng Parquet và lưu xuống HDFS. Ngoài ra, còn có một flow khác để lưu file `danh_sach_sv_de.csv` vào HDFS. Khi truy cập Nifi tại `localhost:8443/nifi`, chúng ta có thể đăng nhập với thông tin định danh như hình trên.

2.1 Đưa template vào Nifi.

Sau khi đăng nhập thành công, ta sẽ import template từ `/nifi_template/DE_Assignment.xml`, trong đó chứa tất cả các processor cần thiết cho nền tảng dữ liệu mà chúng ta đang xây dựng. Tiếp theo, khởi động các controllers/services của processor `ConsumeKafkaRecord` bằng cách truy cập vào phần cài đặt của luồng và phần Controller Services. Khi hoàn thành các bước trên, ta sẽ có 2 luồng như sau:



Hình 4: Các luồng ở trong Nifi.

2.2 Chuyển data vào HDFS.

Thư mục `/nifi_volume` sẽ là nơi mount các file cấu hình của Hadoop vào NiFi cũng như các loại dữ liệu bên ngoài khác. Về phần cấu hình, có hai file quan trọng là `core-site.xml` và `hdfs-site.xml`. Chi tiết nội dung và ý nghĩa của từng trường đã được liệt kê ở source code. Với luồng đầu tiên, để đưa file `danh_sach_sv_de.csv` vào HDFS, cần lưu ý chọn "run once" thay vì "run", vì với cấu hình hiện tại, chúng ta muốn giữ file này ở vị trí ban đầu. Nếu chọn "run", processor sẽ chạy liên tục và lấy vô hạn file. Với luồng thứ hai để chuyển records từ Kafka vào HDFS, có thể chạy theo tuần tự bình thường.

2.3 Một số thông tin cấu hình của Nifi.

Vì Kafka sử dụng Avro để mã hóa dữ liệu, chúng ta cần phải cấu hình NiFi sao cho nó có thể giải mã được chúng. Để đạt được điều này, chúng ta nên sử dụng processor ConsumeKafkaRecord. Thiết lập này yêu cầu hai dịch vụ chính: ConfluentSchemaRegistry và AvroReader. Với ConfluentSchemaRegistry, ta cần cấu hình để nhận diện được Schema Registry của Kafka, có thể được truy cập qua đường dẫn `http://my-schema-registry:8081` như được mô tả ở Hình 1. Đối với

AvroReader, chúng ta chỉ cần kết nối nó với ConfluentSchemaRegistry và đặt strategy là Confluent Content-Encoded Schema Reference. Với những cấu hình này, processor ConsumeKafkaRecord sẽ hoạt động đúng chuẩn với Kafka.

3 Triển khai Hadoop.

```yaml
namenode:
  image: bde2020/hadoop-namenode:2.0.0-hadoop3.2.1-java8
  container_name: namenode
  ports:
    - 9000:9000
    - 9870:9870
  environment:
    - CLUSTER_NAME=test
  env_file:
    - ./hadoop.env

datanode1:
  image: bde2020/hadoop-datanode:2.0.0-hadoop3.2.1-java8
  container_name: datanode1
  ports:
    - 9864:9864
    - 9866:9866
  depends_on:
    - namenode
  env_file:
    - ./hadoop.env
  environment:
    SERVICE_PRECONDITION: "namenode:9870"
```

Hình 5: Cấu hình của Hadoop.

Với Hadoop, chúng ta chỉ cần triển khai nó với các cấu hình đơn giản, cho phép các dịch vụ khác tương tác dễ dàng. Trong nền tảng hiện tại, do khối lượng dữ liệu chưa quá lớn, chỉ cần một NameNode và một DataNode là đủ. Để truy cập giao diện web của Hadoop, ta có thể sử dụng địa chỉ `localhost:9870`. Còn để các dịch vụ tương tác với Hadoop, ta dùng địa chỉ `hdfs://namenode:9000`.

4 Triển khai Spark và xử lý dữ liệu với PySpark

Tương tự như Hadoop, việc triển khai Spark cũng được thực hiện hết sức đơn giản. Ở đây, em chọn sử dụng PySpark để có thể sử dụng ngôn ngữ Python vì em cảm thấy quen thuộc với nó. Sau khi triển khai, ta có thể truy cập và làm việc với các notebooks ở địa chỉ `localhost:8888` với mật khẩu là "admin". Để thay đổi mật khẩu, ta chỉ cần tạo một file Python mới và cài đặt thư viện `notebook`. Sau đó, ta sẽ sử dụng hàm `passwd` từ `notebook.auth`. Nhập mật khẩu mới vào hàm này và sao chép kết quả được đưa . Tiếp theo, dán kết quả vào file `docker-compose.yml` ở vị trí tương ứng như trong hình dưới đây là ta đã có mật khẩu mới.

```
spark-notebook:
  image: jupyter/pyspark-notebook
  command: start-notebook.sh --ServerApp.password='sha1:d2887803f
  ports:
    - 8888:8888
    - 4040:4040
  depends_on:
    - namenode
  volumes:
    - ./spark-notebooks:/home/jovyan/work
```

Hình 6: Cấu hình của Spark.

Về các bước xử lý dữ liệu, các đoạn code và output của chúng đều có chi tiết ở link. Sau khi dữ liệu đã được chuyền vào Hadoop, ta chỉ cần truy cập vào notebook và chạy các block từ đầu đến cuối, việc xử lý dữ liệu sẽ được thực thi. Dưới đây em sẽ chỉ đưa ra các bước thực hiện mà em đã làm trong notebook:

1. Sau khi khởi tạo session của PySpark, việc đầu tiên ta làm đó là lấy dữ liệu dưới dạng parquet dưới HDFS lên Spark. Cụ thể, ta cần lấy thông tin của các activity dưới dạng parquet và danh sách sinh viên dưới dạng csv. Với activity, em đặt tên cho dataframe là `activity_df` còn với sinh viên, em đặt là `student_df`.

2. Tiếp đến, em chuyển kiểu dữ liệu của timestamp từ string sang date của `activity_df`, hỗ trợ cho việc truy vấn và sắp xếp dữ liệu ở công đoạn sau.

3. Trong bước tiếp , em thực hiện join 2 dataframe nêu trên để có thể lấy ra những thông tin em cần. Trường `activity_df.studentCode` và `student_df.id` đã được chọn làm khoá để join. Ngoài ra em cũng lấy thêm cả trường `student_df.name`

4. Sau đó, em chạy lệnh `groupBy` với trường `activity` và `timestamp` từ dataframe đã được join ở trên. Khi thực hiện bước này, em có thể tính tổng số file theo kiểu mà sinh viên đó tương tác theo từng ngày.

5. Với dataframe trên, việc xử lý dữ liệu đã được hoàn tất và đã sẵn sàng cho việc lưu trữ xuống HDFS cũng như một bản ở máy local. Đây là ví dụ cho file output được đưa ra sau khi xử lý dữ liệu

timestamp	studentCode	student_name	activity	totalFiles
20240610	37	Đào Anh Vũ	execute	9
20240610	37	Đào Anh Vũ	read	8
20240610	37	Đào Anh Vũ	write	5
20240611	37	Đào Anh Vũ	read	10
20240611	37	Đào Anh Vũ	write	7
20240612	37	Đào Anh Vũ	execute	9

Bảng 1: File output được lưu trữ dưới dạng CSV.