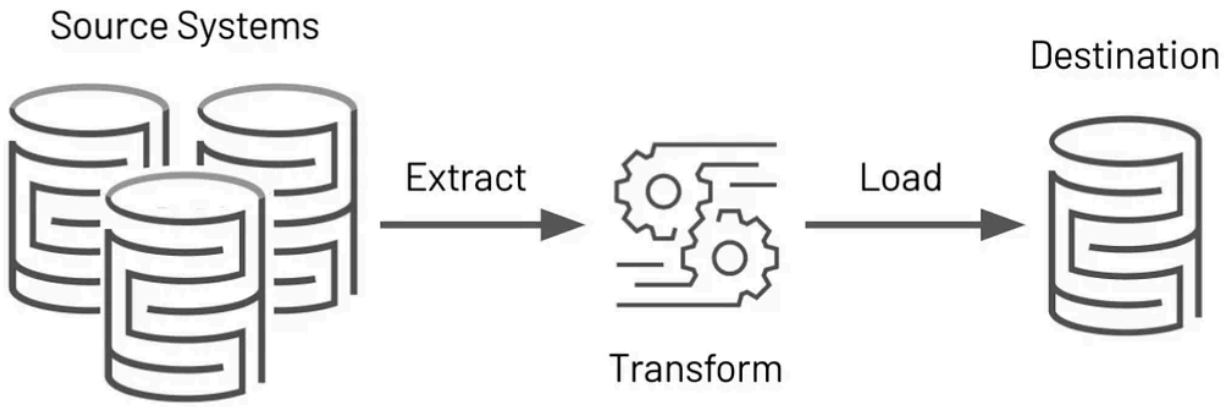
Real-Time ETL with Java & Groovy

6 min read · Dec 30, 2021

israel mwangoka Follow

Listen Share More

ETL*(Extract, Transform, Load)* is a technique that involves **E**xtraction of data from sources like databases, files, web APIs etc. **T**ransformation of the extracted data to meet the business needs & **L**oading the data into destinations like Data Warehouse, Data Lake etc. The technique has more than 40 years but there are other variants too like ***ELT*** where data is extracted ,loaded into databases and get transformed using SQL, scripts etc.



ETL process

There are number of implementations for this technique but here we'll group them into either Batch ETL or Real-Time ETL.

Batch ETL: Means data is being extracted from sources after particular time intervals, processed and load it into destinations i.e. data is processed in batches.

Real-Time ETL: This is probably the latest ETL technology in which data is processed immediately upon their arrival. Sometimes this is called Stream processing or Real-Time processing.

Batch ETL still have use cases but it is slowly dying. It is being replaced by Near Real-time ETL or Real-time/Stream ETL. In this post will focus on Real-time ETL.

Basically in this post we'll deal with '**T**' see how we can manipulate data received from several sources applying business logic and finally sending them to the destinations.

Apache Kafka

As you know, when a new technology evolves always comes with new set of tools well. Today when you mention stream processing somewhere you will mention Apache Kafka as well. Kafka is used by big companies like Uber, LinkedIn, Airbnb, Yahoo etc. and most of them use it for real-time processing of data. Due to this, Kafka has many competitors especially from traditional messaging based solutions like RabbitMQ, NATS, Redis etc. which have added RabbitMQ streams, Jet Strems & Redis Streams(v5 or higher) respectively to support stream processing.

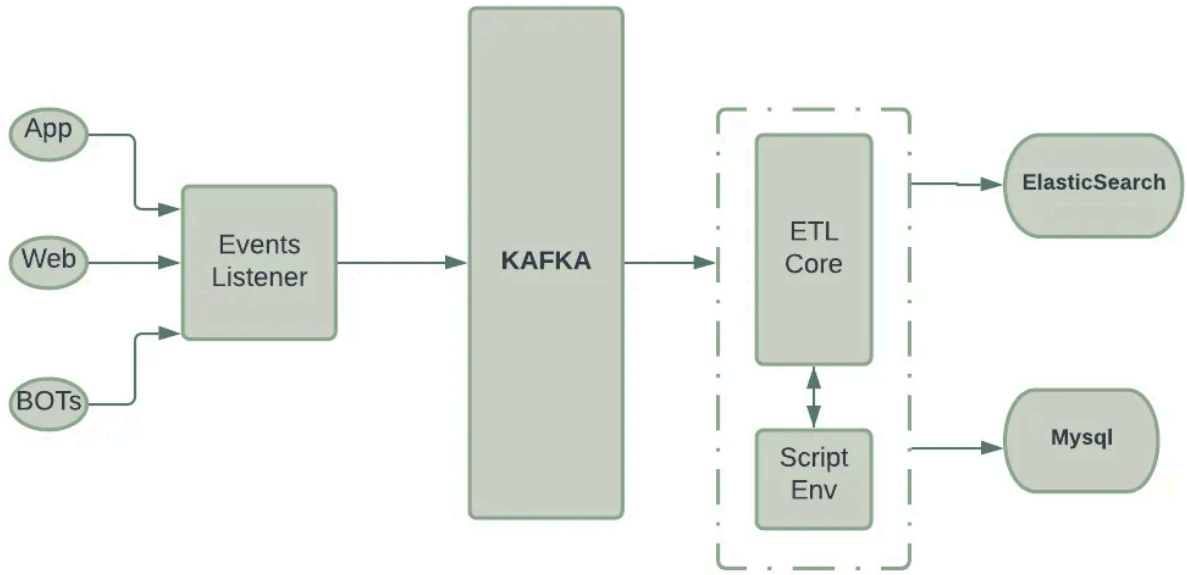
We'll use Kafka *(Compare it with Apache Pulsar)* as our streaming platform in which data will be published from channels to Kafka topic then consumed from the topic, manipulated and finally send it to defined destination(s). Real-time ETL can be used in payment systems, fraud detection, user activity trackers etc. In this post we'll see how we can use real-time ETL with Java & Groovy for data transformation/manipulation before sending to data warehouse(MySQL) & Elasticsearch for indexing. In this post we'll not use any Kafka connector(s) i.e. we'll be sending to destinations directly from groovy script.

Problem Statement

In many business entities like Banks, MNOs, e-Commerce etc. normally have products & services which their customers can access through channels like Web Apps, Mobile Apps, Chatbots, IVRs etc. Business analysts/Product owners at some point they will need to understand the performance of their products & services from each channel that they have. If you have few channels you can simply collect data from individual channel which is normally not the case. So, an efficient solution is needed that can aggregate those information from different channels and

send to a certain platform that business analyst/any can easily use the data in real-time. In our case will assume the analyst can use data from either Elasticsearch or MySQL.

Our Architecture



Real-time ETL flow diagram

Our application will have 2 microservices:

Events Listener that essentially listens to http requests from sources(channels) and publish (Kafka producer) to Kafka topic *ChannelUsersTopic*. The event will have this JSON format:

```
{
    "customer_id": "1230006",
    "event_source_id": "POS",
    "event_time": "2021-11-20 12:00:11",
    "event_type": "BUY_DSTV_PACKAGE_500",
    "customer_name": "Jefembe Mbefeje"
}
```

Event JSON format

And our event REST controller will look like this:

```java
@Slf4j
@RestController
public class EventsController {

    @Value("${kafka.event.topic-name}")
    private String topicName;

    private KafkaTemplate<String, String> kafkaTemplate;

    public EventsController(KafkaTemplate<String, String> kafkaTemplate) {
        this.kafkaTemplate = kafkaTemplate;
    }

    @PostMapping("/event")
    public ResponseEntity<String> eventsHandler(@RequestBody String event){
        log.info("Received event from channel: {}", event);
        kafkaTemplate.send(topicName, event);
        return new ResponseEntity<>(HttpStatus.ACCEPTED);
    }
}
```

***ETL Core** w*hich is a Kafka consumer that reads events from the Kafka topic and calls the required groovy script from the *scripting repository* to perform required business logic. The implementation supports multiple topics, so if you have a new topic to listen on, just add the topic in *application.properties* file and put the script file into the scripting repository.

The core is performing logics that can rarely change e.g. reading from Kafka topics. Also the core can share its state/data through our famous groovy object called Binding. With Binding you can not only share simple data types between java and groovy but complex types like Objects, Maps, JSON objects etc. In our case we'll share logger and JSON string messages from the Kafka topics.

```java
@Service
@Slf4j
public class ScriptingService {

    @Value("${scripts.dir}")
    private String scriptDir;
    public void scriptHandler(String request, String scriptName){
        try {
            Binding binding = new Binding();
            binding.setVariable( name: "request", request);
            binding.setVariable( name: "log", log);
            new GroovyScriptEngine(scriptDir).run( scriptName: scriptName + ".groovy", binding);
        }catch (Exception e){
            log.error("Failed to process the request: {}", e.getMessage());
        }
    }
}
```

ScriptingService.java

Please note, this is a real world example based on what we learned in my old post here, please go and read if you want to understand the Java-Groovy combo.

Why groovy in ETL?

Groovy is richer in functions, simple syntax moreover you can apply business logic changes at runtime without disturbing your core functions through Groovy Script Engine(GSE). Generally, scripting is good in data transformations even RudderStack, use JavaScript in their c*ustomer data pipeline(CDP)* to create complex scripts that can be applied to events or batches extracted from sources.

Scripting Repository

This is where our scripts that hold specific business logic resides. It is a normal directory which is best to be outside your class-path to leverage the power of GSE. For the purpose of this I'll just put under /resources directory and it will contain only one script called *ChannelUsersTopic.Groovy*. The script will do parse the JSON string received from core, extract the parameters and send to MySQL database and Elasticsearch through its Rest API. In the script you can enrich your data before sending to the required destination(s).

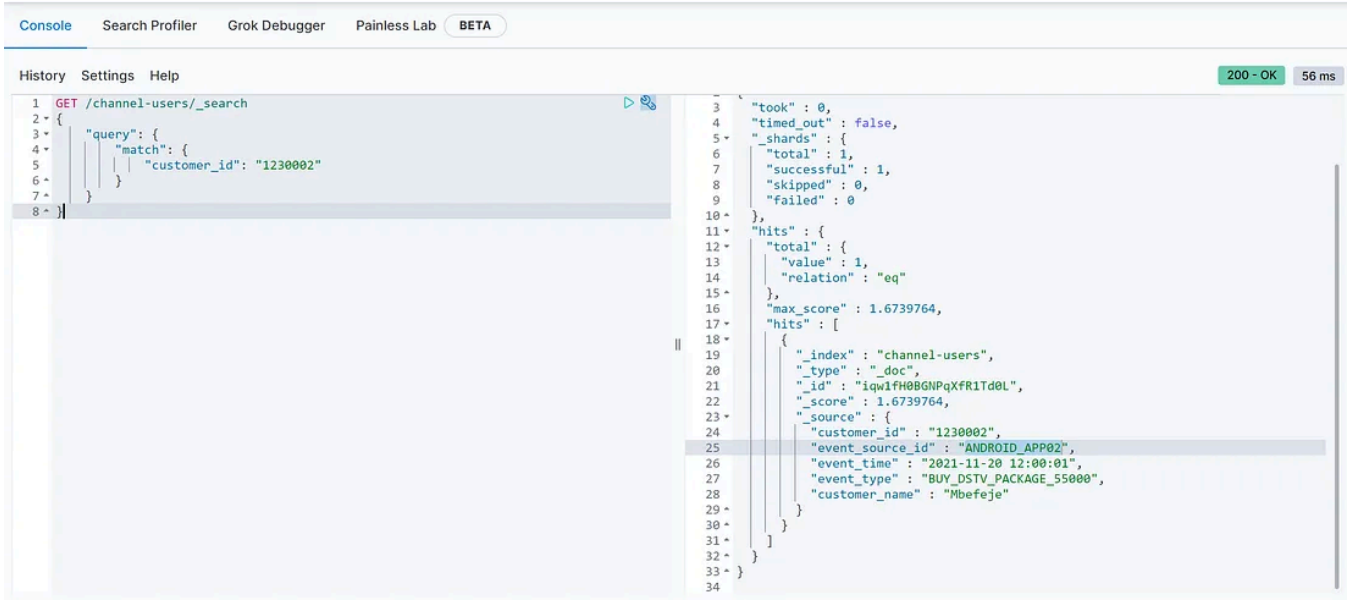
```
try {
    def mysqlUrl = "jdbc:mysql://localhost:3306/channel_reports"
    def userName = "root"
    def password = "root"
    def sql = Sql.newInstance(mysqlUrl, userName, password)
    sql.executeInsert( gstring: """insert into channel_users values(${requestMap.customer_id},
                            ${requestMap.event_source_id},
                            ${requestMap.event_time},
                            ${requestMap.event_type},
                            ${requestMap.customer_name})""")
    sql.updateCount > 0 ? log.info("Data saved successfully into mysql: ${sql.updateCount}") : log.error("Failed t
}catch(Exception e){
    log.error("Failed to save data into mysql database: ${e.getMessage()}")
}

try{
    def elasticUrl = "http://localhost:9200/channel-users/_doc"
    def client = new RESTClient(elasticUrl)

    client.request(Method.POST) {
        requestContentType = ContentType.JSON
        body = requestMap
        response.success = { resp -> log.info("Data save successfully to elasticSearch: ${resp.status}")}
        response.failure = { resp -> log.info("Failed to save data to elasticSearch: ${resp.status}")}
    }
}catch(Exception e){
    log.error("Failed to save data into elasticSearch due to: ${e.getMessage()}")
}
```

ChannelUsersTopic.Groovy

My assumption you have a setup already for **MYSQL** and **Elastic stack.** After starting your two instances (listener and core), start publishing events to the events endpoint *http://localhost:8080/event* and check your data in **mysql** and **kibana.**



Event on Kibana

```
MariaDB [channel_reports]> select * from channel_users where customer_id = '1230002';
+-------------+-----------------+---------------------+-------------------------+---------------+
| customer_id | event_source_id | event_time          | event_type              | customer_name |
+-------------+-----------------+---------------------+-------------------------+---------------+
| 1230002     | ANDROID_APP02   | 2021-11-20 12:00:01 | BUY_DSTV_PACKAGE_55000 | Mbefeje       |
+-------------+-----------------+---------------------+-------------------------+---------------+
1 row in set (0.001 sec)
```

Event on MYSQL

Wrapping UP

We have just seen how we can real time process data in ETL pipeline using Java & Groovy script. Within the script you can perform any complex business logics before sending to destinations.

Normally you will use Kafka connectors to communicate between Kafka and MYSQL or Kafka and Elasticsearch though it is not mandatory. With connectors setup, after data manipulation, you will again publish the enriched/formatted data into destination topics that MYSQL & Elasticsearch listens.

Please check all the source codes in my GitHub account below, if you have any issue/advice please let me know in the response section.

medium/etl-core at master · mwangox/medium

Contribute to mwangox/medium development by creating an account on GitHub.

github.com

Happy Learning! Happy New Year!

Kafka Etl Etl Pipeline Groovy Java

Follow

Written by israel mwangoka

59 followers · 93 following

Computer Tech. Enthusiast