KafkaToS3WithPaquet

DataLake Kafka Parquet

Kafka Json S3(Paquet) Flow .

MSA ~

 .

Kafka S3 ,

 Flow Stream

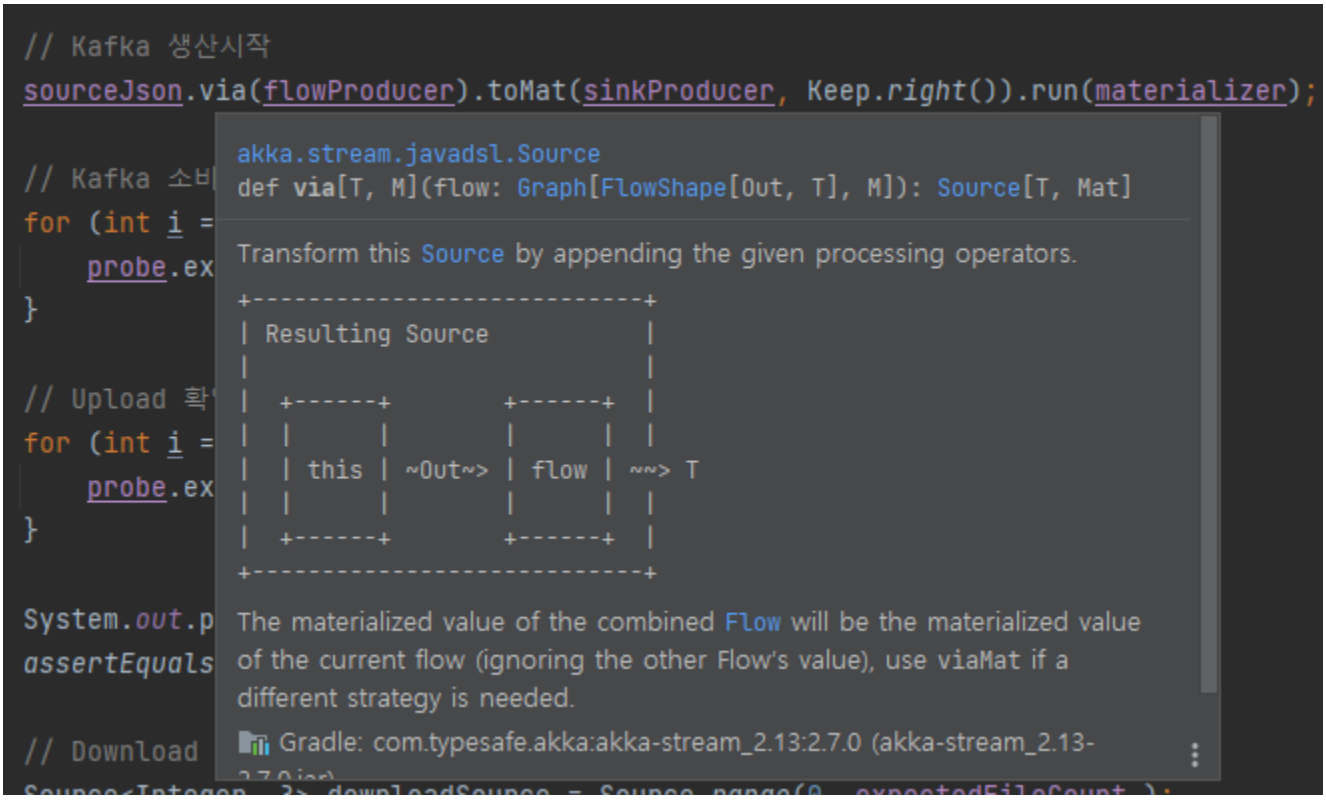
```
    /* ProduceAndConsumeToS3AreOK

+--------------+        +--------------+       +--------------+      +--------------+     +--------------+
|              |        |              |       |              |      |              |     |              |
|  Kafka       |        |  Kafka       |       |   Flow       |      | S3 Upload    |     | S3 DownLoad  |
|  (Producer)  |------->|  (Consumer)  |------>|   Convert    |----->|              |---->|              |
|              |        |              |       |   Pqrquet    |      |              |     |              |
|              |        |              |       |              |      |              |     |              |
+--------------+        +--------------+       +--------------+      +--------------+     +--------------+

  JSON  N   ->        ->     N    ->    N   ->  N

     */
```

AKKA Stream Flow



AKKA Stream .

- Source : .
- Via : .
- FlowSome : .
- Sink : , . Sink , , , . , , Sink .
- run : Sink , run . AkkaSystem Dispacher .

Kafka JSON

```
Source<String, NotUsed> sourceJson = Source.range(1, testCount)
        .map(number -> {
                // JSON
                S3TestJsonModel s3TestJsonModel = new S3TestJsonModel();
                s3TestJsonModel.count = number;
                String jsonOriginData = mapper.writeValueAsString(s3TestJsonModel);

                //
                S3TestModel s3TestModel = new S3TestModel();
                s3TestModel.jsonValue = jsonOriginData;
                return mapper.writeValueAsString(s3TestModel);
        });

Flow<String, ProducerRecord<String, String>, NotUsed> flowProducer = Flow.of(String.class)
        .map(value -> new ProducerRecord<>(testTopicName, testKey, value));

Sink<ProducerRecord<String, String>, CompletionStage<Done>> sinkProducer = Producer.plainSink(producerSettings);

sourceJson.via(flowProducer).toMat(sinkProducer, Keep.right()).run(materializer);
```

JSON .

.

- JSON
  - jsonserialize (.)

JSON

```
public class S3TestModel {

    public String name = "S3TestModel";

    public String version = "0.0.1";

    public String jsonValue;
}

public class S3TestJsonChildModel {

    public String name = "S3TestJsonChildModel";

    public int count = 2;

    public String subData = "SomeData";
}
```

- , json json .
- .
  - N . .
    - parquet .
    - , .
      - , (NearRealTime) .

Kafka JSON S3 Parquet

```
Consumer
        .plainSource(
                consumerSettings,
                Subscriptions.topics(testTopicName))
        .grouped(maxEntityPerFile)
        .to(Sink.foreach(group -> {

                S3TestModel model = new S3TestModel();
                model.jsonValue = "example data";

                Schema schema = new Schema.Parser().parse(schemaString);

                // Avro GenericRecord
                GenericRecord record = new GenericData.Record(schema);

                curFileIdx[0]++;

                String dynamicFileKey = fileName + curFileIdx[0];

                group.forEach(msg -> {
                try {
                        // AnyJson  ~ S3TestModel
                        S3TestModel obj = mapper.readValue(msg.value(), S3TestModel.class);
                        record.put("name", obj.name);
                        record.put("jsonValue", obj.jsonValue);

                } catch (JsonProcessingException e) {
                        throw new RuntimeException(e);
                }
                debugKafkaMsgAndConfirm(msg.key(), msg.value(), confirmActor, testKey, "consumer1");
                });
        .................
```

- , maxEntityPerFile .
  - maxEntityPerFile Entity GenericRecord . StreamAPI .
- GenericRecord json Parquet .

S3

```
Source.single(byteString)
        .runWith(S3.multipartUpload(bucketName, dynamicFileKey)
                .withAttributes(S3Attributes.settings(s3Settings)), materializer)
        .thenAccept(result -> {
                LocalTime now = LocalTime.now();
                DateTimeFormatter formatter = DateTimeFormatter.ofPattern("HH:mm:ss");
                String formatedNow = now.format(formatter);
                System.out.println(formatedNow + " Upload complete: " + result.location());
                confirmActor.tell("s3UploadOK", null);
        })
        .exceptionally(throwable -> {
                System.err.println("Upload failed: " + throwable.getMessage());
                return null;
        });
```

- parquet byte s3 .
- dynamicFileKey : .

S3

```
downloadSource
        .runForeach(index -> {
        String dynamicFileKey = fileName + (index + 1);

        // S3
        CompletionStage<Optional<Pair<Source<ByteString, NotUsed>, ObjectMetadata>>> download =
                S3.download(bucketName, dynamicFileKey)
                        .withAttributes(S3Attributes.settings(s3Settings))
                        .runWith(Sink.head(), materializer);

        download.thenAccept(optionalSourcePair -> {
                        optionalSourcePair.ifPresent(sourcePair -> {
                        LocalTime now = LocalTime.now();
                        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("HH:mm:ss");
                        String formatedNow = now.format(formatter);
                        Source<ByteString, ?> downloadbyteSource = sourcePair.first();
                        downloadbyteSource.runWith(Sink.foreach(byteString -> System.out.println(formatedNow +
" Downloaded: " + dynamicFileKey)), materializer);
                        confirmActor.tell("s3DownloadOK", null);
                        });
                })
                .exceptionally(throwable -> {
                        System.err.println("Download failed for " + dynamicFileKey + ": " + throwable.
getMessage());
                        return null;
                });
        }, materializer);
```

- .
  - Parquet .

```
// KAFKA 1      ~
if (testKey.equals(key)) confirmActor.tell("kafkaOK", null);

=================   Context =================

// Kafka          ,       ~
//
for (int i = 0; i < testCount; i++) {
 probe.expectMsg(Duration.ofSeconds(5), "kafkaOK");
}
```

.

.

- .

Flow

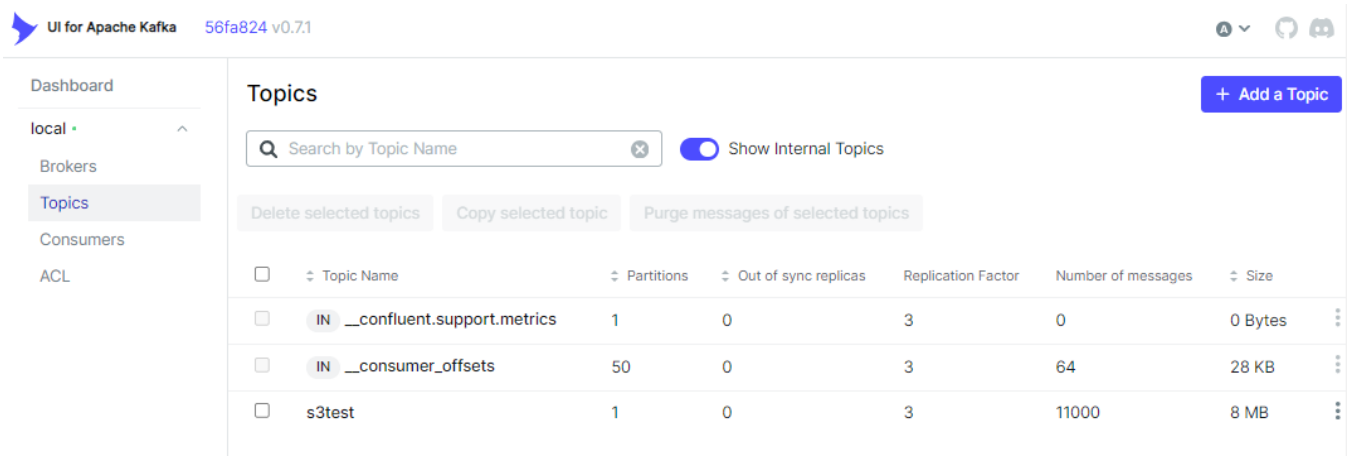
MockServer() Test

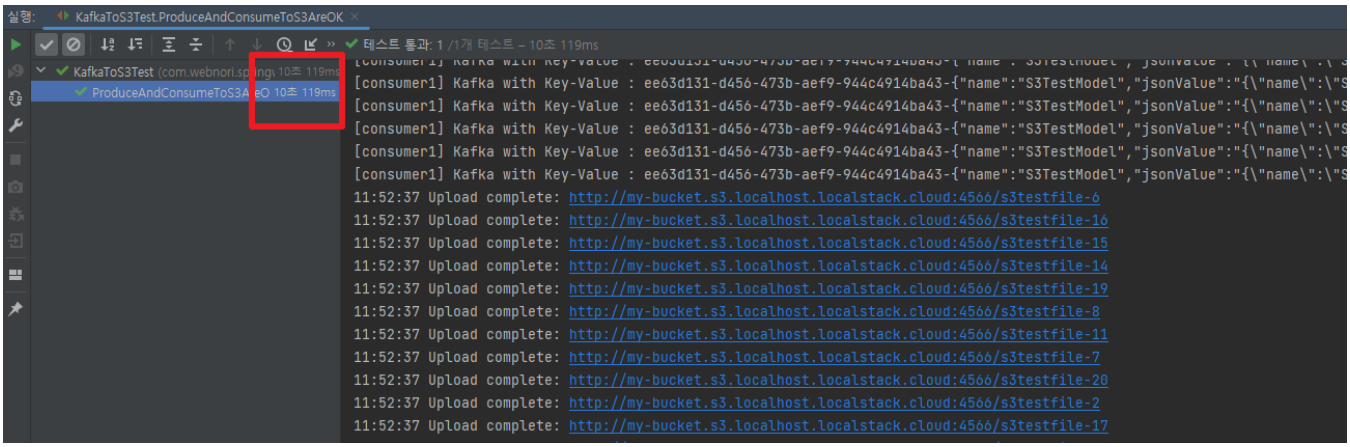
Kafka Topic .

Flow (MSA)

Flow .

~ **1.**





- .
  - , GraceFulDownn .

Kafka S3 Flow ~ .

alphaka .

.

```
  retry-settings {
   max-retries = 3
   min-backoff = 200ms
   max-backoff = 10s
   random-factor = 0.0
  }
```

.

test.conf

```
# Properties for akka.kafka.ProducerSettings can be
# defined in this section or a configuration section with
# the same layout.
akka.kafka.producer {
  # Config path of Akka Discovery method
  # "akka.discovery" to use the Akka Discovery method configured for the ActorSystem
  discovery-method = akka.discovery

  # Set a service name for use with Akka Discovery
  # https://doc.akka.io/docs/alpakka-kafka/current/discovery.html
  service-name = ""

  # Timeout for getting a reply from the discovery-method lookup
  resolve-timeout = 3 seconds

  # Tuning parameter of how many sends that can run in parallel.
  # In 2.0.0: changed the default from 100 to 10000
  parallelism = 10000

  # Duration to wait for `KafkaProducer.close` to finish.
  close-timeout = 60s

  # Call `KafkaProducer.close` when the stream is shutdown. This is important to override to false
  # when the producer instance is shared across multiple producer stages.
  close-on-producer-stop = true

  # Fully qualified config path which holds the dispatcher configuration
  # to be used by the producer stages. Some blocking may occur.
  # When this value is empty, the dispatcher configured for the stream
  # will be used.
  use-dispatcher = "akka.kafka.default-dispatcher"

  # The time interval to commit a transaction when using the `Transactional.sink` or `Transactional.flow`
  # for exactly-once-semantics processing.
  eos-commit-interval = 100ms

  # Properties defined by org.apache.kafka.clients.producer.ProducerConfig
  # can be defined in this configuration section.
  kafka-clients {
  }
}

akka.kafka.consumer {

  enable.auto.commit = true

  kafka-clients {
    bootstrap.servers = "localhost:9092"
  }
}

akka.kafka.committer {

  # Maximum number of messages in a single commit batch
  max-batch = 1000

  # Maximum interval between commits
  max-interval = 3s

  # Parallelsim for async committing
```

```
  parallelism = 100

  # API may change.
  # Delivery of commits to the internal actor
  # WaitForAck: Expect replies for commits, and backpressure the stream if replies do not arrive.
  # SendAndForget: Send off commits to the internal actor without expecting replies (experimental feature since
1.1)
  delivery = WaitForAck

  # API may change.
  # Controls when a `Committable` message is queued to be committed.
  # OffsetFirstObserved: When the offset of a message has been successfully produced.
  # NextOffsetObserved: When the next offset is observed.
  when = OffsetFirstObserved
}

#     LocalStack , AWS-S3     .
# link : https://github.com/akka/alpakka/blob/main/s3/src/main/resources/reference.conf
alpakka.s3 {
  buffer = "memory"
  disk-buffer-path = ""

  # default values for AWS configuration
  aws {
    credentials {
      provider = static
      access-key-id = "test"
      secret-access-key = "test"
    }

    region {
      provider = static
      default-region = "us-east-1"
    }
  }

  path-style-access = true
  access-style = virtual
  endpoint-url = "http://localhost:4567"
  list-bucket-api-version = 2
  validate-object-key = true

  retry-settings {
    max-retries = 3
    min-backoff = 200ms
    max-backoff = 10s
    random-factor = 0.0
  }

  multipart-upload {
    retry-settings = ${alpakka.s3.retry-settings}
  }

  sign-anonymous-requests = true
}
```

- kafka / s3 (Flow) .
  - kafka config kafka .
  - Flow .
    - Flow .

AkkaStreamAPI

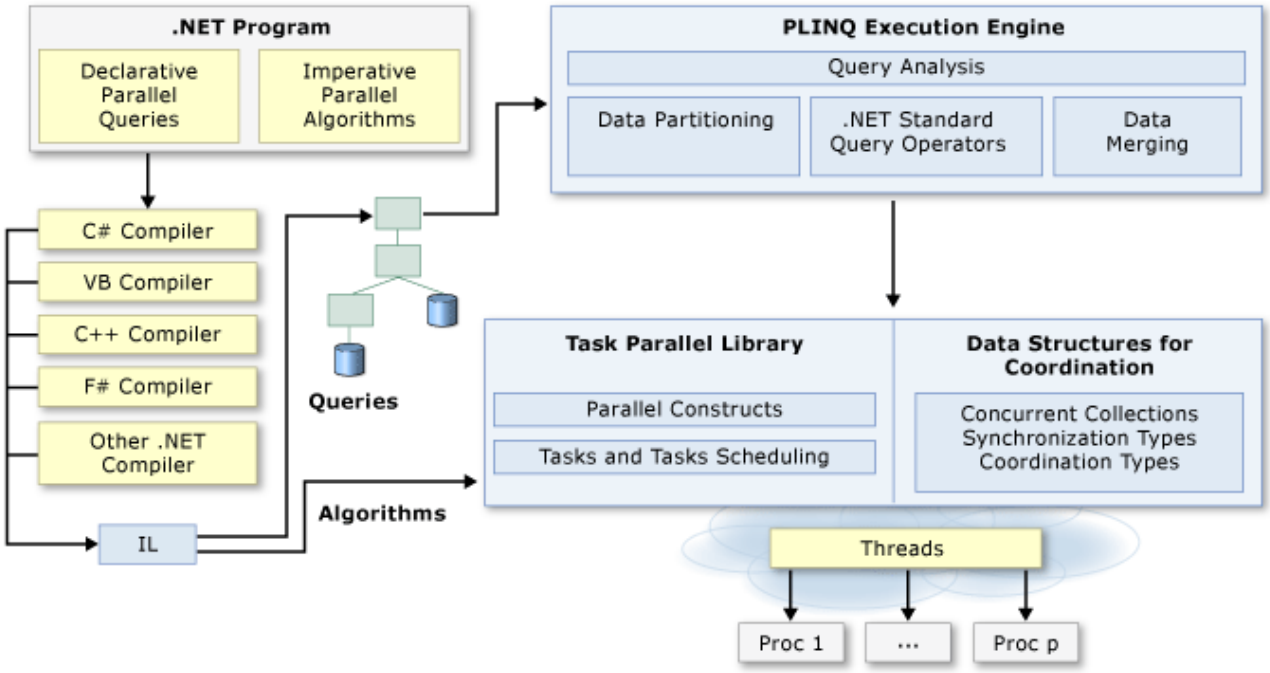
.

:

- https://github.com/psmon/java-labs/blob/master/springweb/src/test/java/com/webnori/springweb/alpakka/reactive/KafkaToAnyTest.java
- https://github.com/psmon/java-labs/blob/master/infra/REAMME.md

Stream **AkkaStream** .

StreamAPI .



- Webplux/rx.Java **ReactiveStream** .
- Linq/TPL/Await .

```
Erik Meijer ~

",  , C#,,   PHP async, await  .     .

    ?   .  .  9

asymc, await .    "
```

- await completedFuture .

- https://www.baeldung.com/java-9-stream-api
- https://docs.spring.io/spring-framework/reference/overview.html
- https://reactivex.io/
  - Java, Scala, C#, C++, Clojure, JavaScript, Python, Groovy, JRuby .
- https://learn.microsoft.com/ko-kr/dotnet/standard/parallel-programming/

- Future and Promise

AkkaStream .

.

~ **?** .

.

:

- 00.UnitTest -
- https://github.com/psmon/NetCoreLabs - .

: alpakka