Candidate test
Review

16 March 2021

[Restricted–External]

MediaMath

Problem statement / task

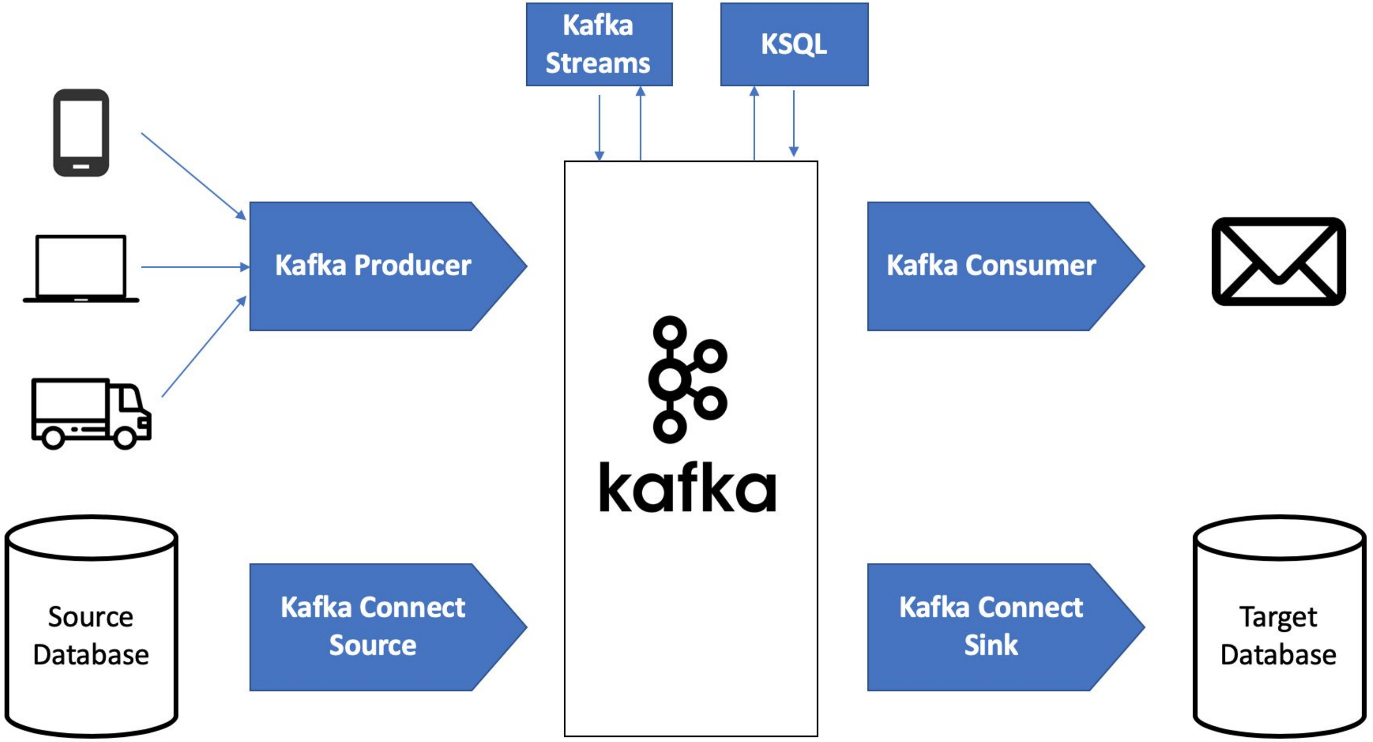
MediaMath task

Project definition / criterias

- Reading files from the filesystem when they arrive
- Parsing the data in the files
- Serializing that data in preparation for publishing to Kafka clusters
- Creating a Kafka client
- Publishing to multiple internal clusters
- Publishing to a Kafka cluster running on a cloud platform
- Removing the files from the filesystem when their contents have been published
- Providing an SDK for use by other teams in your startup for some of the key, high-level functionality

Major decisions / choices

Kafka Connect



Solution 1#

Using Kafka Connect

Kafka Connect allows various source and sink connectors to be installed, making the task relatively simple.

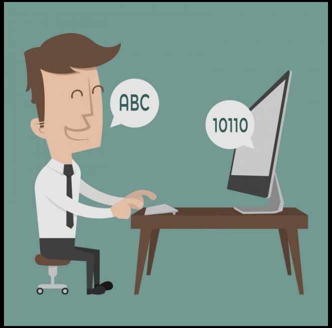
Not to mention transforming the data via SMT-s is relatively simple to implement too.

Since it can be deployed in both standalone and distributed fashion, i think it could work in this scenario too.

Only caveat is that it won’t really show any coding much rather just configuration options in general

Solution 1#

Using Kafka Connect



For non-technical audience

For non-engineers it means, there are various plugins available for Kafka connect for both **input** and **output**.

You can mix and match these based on your preference. For example you can have a file system watcher and a database source which can publishes messages to your Kafka cluster if changes are detected.

Similarly on the “Sink” side, you can have your internal Kafka cluster as well as a Cloud service, other databases, files and so on.

This requires minimal coding on the backend, most of it just configuration, hence it is not really a good way to demonstrate coding skills in a task like this.

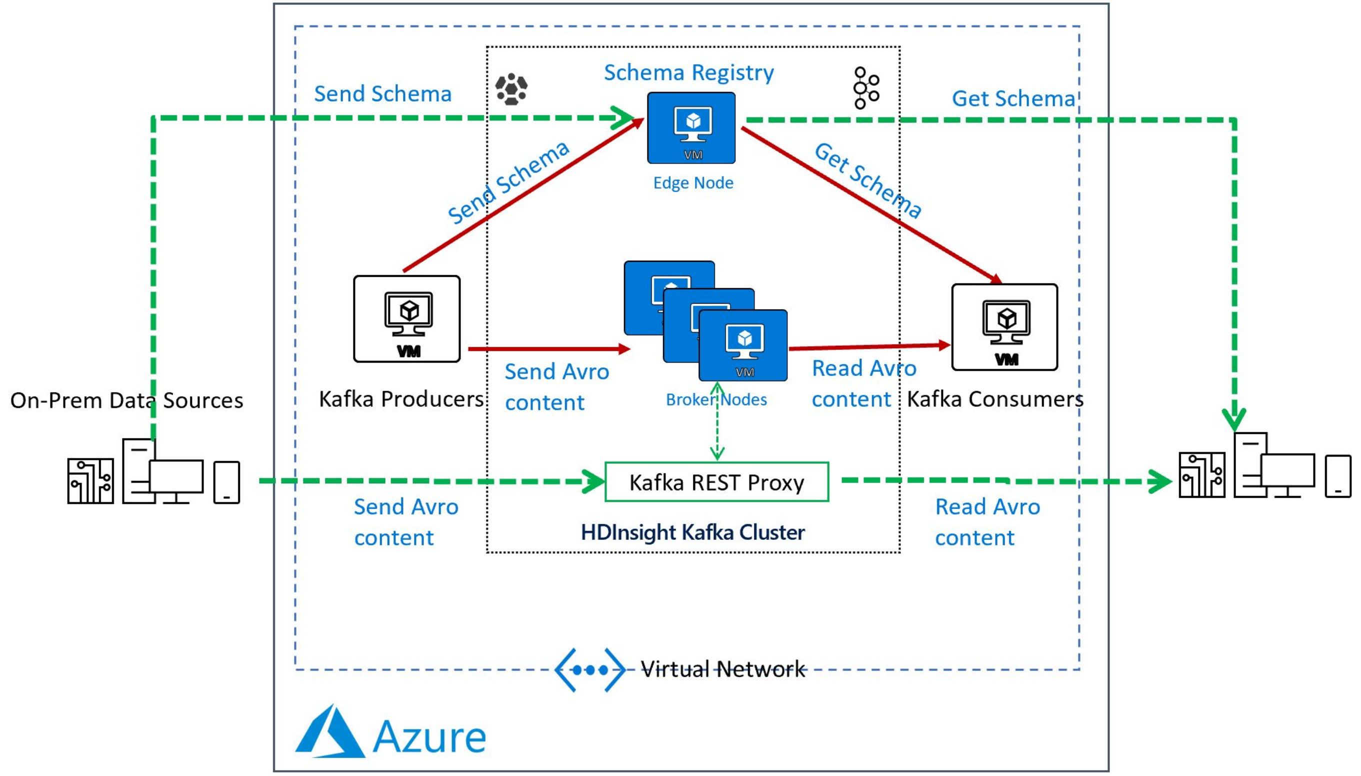
Solution 2#

Using custom Consumers and Producers

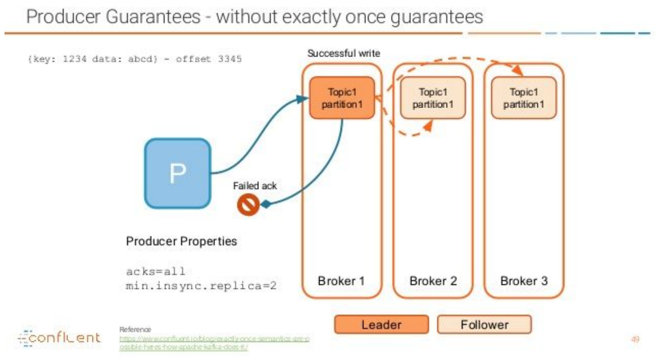
A more expected solution would be to use a traditional consumer and 2 producers (1 which publishes to the internal Kafka cluster and 1 which communicates with ccloud / or any other platform solution)

Wasn't entirely sure how much of a bottleneck it could cost to introduce an Avro based schema registry solution where now before each message consumption/production the reference id needs to be resolved from the in-memory Registry service via REST API. Although the registry does store these in a fault tolerant way in topics but without a proper benchmark I am uncertain of the implications of MediaMath might have to sustain.

Kafka Schema Registry



Task: Removing the files from the FS after publish


Producer Guarantees - without exactly once guarantees

```
{key: 1234 data: abcd} - offset 3345
```

Producer Properties

```
acks=all
min.insync.replica=2
```

Reference
https://www.confluent.io/blog/exactly-once-semantics-are-possible-heres-how-apache-kafka-does-it/

How to implement it?
By leveraging the channel returned from publishing a message, we can listen for the final acknowledgement to make sure we only trigger a delete on our filesystem once its guaranteed to be replicated on multiple topics/clusters

Why is it important?
Brokers can go offline at any time, if the message is not yet written to the followers and the leader goes down, the message can get lost, because zookeeper will elect a new leader who don't know about this message.

An ideal approach

Contracts and modules

- Both consumers and producers are disconnect from the concept of Kafka
  -> These should be modules in my opinions, using standard input/out either using some sort of worker pool management or internal channels.
  -> Should be easily mockable and swappable for testing and more robust A/B testing with feature flags.

- Enforced data pipeline is something really important. Data evolve over time hence a schema which support validation is a must. (Avro, Protobuf, Thrift)
  -> Unrecognised messages can be dropped into a Dead-Letter Queue

- Fast data processing. Each data type / steps should be handled by separate services.
  (Like Kafka stream processors)

An ideal approach

Contracts and modules

- Both consumers and producers are disconnect from the concept of Kafka
  -> These should be modules in my opinions, using standard input/out either using some sort of worker pool management or internal channels.
  -> Should be easily mockable and swappable for testing and more robust A/B testing with feature flags.

- Enforced data pipeline is something really important. Data evolve over time hence a schema which support validation is a must. (Avro, Protobuf, Thrift)
  -> Unrecognised messages can be dropped into a Dead-Letter Queue

- Fast data processing. Each data type / steps should be handled by separate services.
  (Like Kafka stream processors)

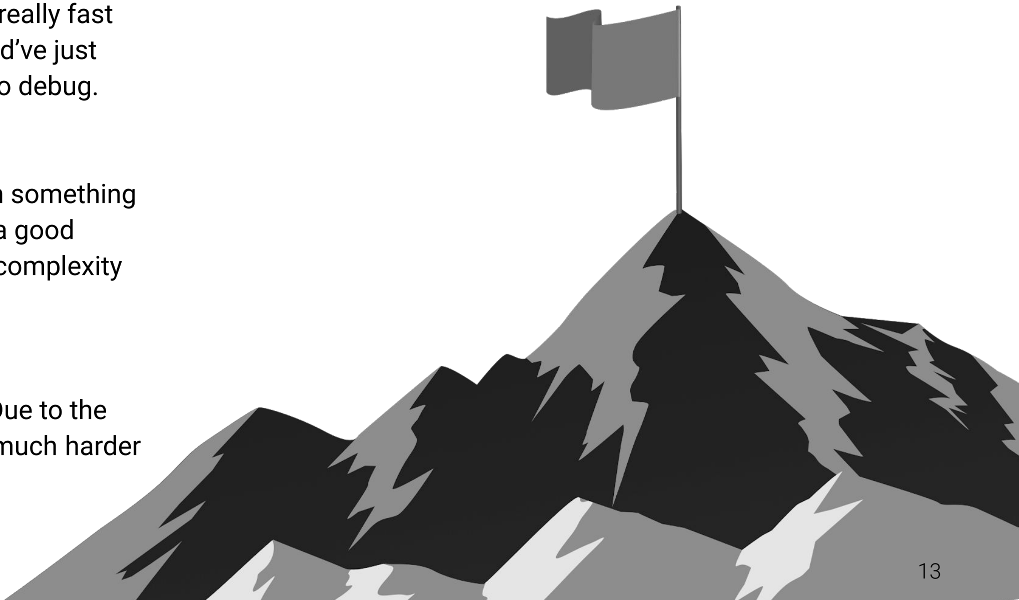
My notes;

CCloud
Had some issues with the authentication on the CCloud. There are probably some discrepancies in their documentation or the Go implementation just a bit different. Haven't spend too much time on figuring out.

Mock generation
I have noticed that the Avro mock generator sometimes triggers double events on the FS filewatcher. This could be because the watcher time interval is really fast and picks up the file creation and alteration as a separate event. I could've just dropped a hard-coded file but it would've been too boring and harder to debug.

Go implementation
Some of these libraries lack a lot of things, ideally I would have written something myself. For tracing its necessary to pass the context around which is a good practice to begin with anyway, but the bigger ones hiding most of the complexity from the user, making it harder to customise.

Testing
I like interchangeable elements and decoupling above anything else. Due to the weird contracts provided by the libraries I mentioned above I found it much harder to test it neatly / properly.



Thank you

Tamas Feik