**Proof of concept – Event hub**

# Background

This proof of concept (PoC) was ordered to verify if Apache Kafka could be a candidate as the technical solution of the event management parts of the API-management platform.

The purpose of the PoC is to prove integration patterns using an Event Hub based on meaningful/representative use cases for IKEA. The work was run at IKEA IT during 2018 March to May by Harri Willstrand, Niklas Bergkvist IKEA IT and Luis Weir, James Neate Cap Gemini UK.

# Executive Summary

This was a Proof of concept run at IKEA 2018 to look in to the software product Kafka. The PoC has found that Kafka should be tried out in real project but also need to be explored in more areas.

# Objectives

It is expected that the following outcomes will reached:

* Complexity involved in implementing the Event Hub and the different integration scenarios chosen
* Estimated complexity to operate the solution (e.g. transaction monitoring, exception handling, etc).
* Estimated throughput (tps) that can be supported for multiple message sizes (e.g. STMS large message sizes)
  + From 5KB to 150 MB message sizes
  + Explore how VETRO or equivalent could be achieved
  + In terms of throughput, target 300k / day to millions a day
* Viability for supporting the main horizontal integration patterns: batch/ETL, pub/sub
* Steps executed to complete the scenarios
* Documentation of issues/challenges faced during the implementation
* A recommendation on whether an Event Hub solution is a suitable for IKEA to support the aforementioned patterns

# Approach

The PoC was initially to run three different scenarios. The 3rd scenario was not as critical, and was taken out of scope. Instead an additional test on how Kafka components interaction and behaviour was added. This to understand the infrastructure implications, especially from a receiving organisation point of view (operations).

The way to set up Kafka was to use off the shelf configuration as much as possible, not optimizing for any special use-case.

### Scenario 1 – Simple Integration via Event Hub

## 

1. A message is posted in JSON format to the Topic via the Kafka Rest proxy using Apache Benchmark
2. A consumer app calls the Kafka Rest Proxy to access the latest records in the topic

The first scenario focus on big messages in various sizes sent from one provider to one consumer using the Kafka REST proxy. The purpose of the test is to monitor throughput both for posting and consuming app.

No optimization was made on the Kafka set up. The only configuration changes made was to enable bigger message size than default configuration.

#### Producer test - Scenario 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test ID 1 | Concurrent Threads | Requests Per Thread | Message Size (MB) | Total Messages Sent | Total Message Size Transferred (GB) | Duration |
| 1 | 1 | 1 000 | 10,9 | 1000 | 10,64 | 1m 52s |
| 2 | 1 | 100 000 | 10,9 | 100 000 | 1064,45 | 185m 0s |
| 3 | 10 | 1 000 | 10,9 | 10 000 | 106,45 | 11m 33s |
| 4 | 10 | 10 000 | 10,9 | 100 000 | 1064,45 | 103m 7s |
| 5 | 1 | 1 | 101,8 | 1 | 0,10 | 3.5 s |
| 6 | 1 | 10 | 101,8 | 10 | 0,99 | 67.02s |
| 7 | 10 | 1 | 101,8 | 10 | 0,99 | 42.86s |
| 8 | 10 | 10 | 101,8 | 100 | 9,94 | 5m 57s |
| 9 | 10 | 100 | 101,8 | 1 000 | 99,41 | 58m 40s |

The test was run using variation on message sizes, concurrent threads posting and the number of messages sent. First the producer test was run. After that when messages was stored at Kafka instance, the consuming test was run.

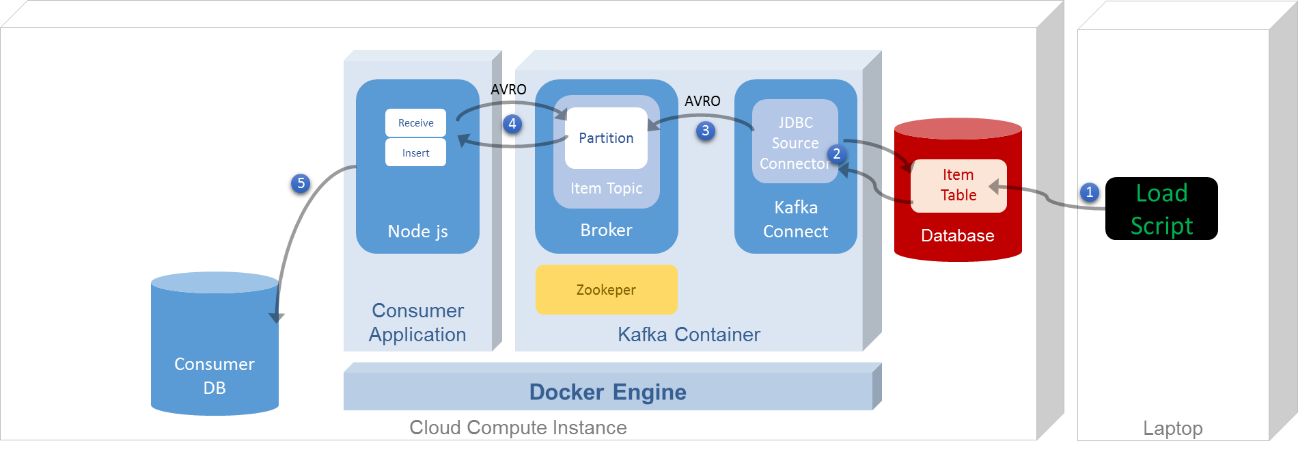
Message size has impact on throughput, 1000 messages of 10,9 MB each was taking 1/3 of the time of sending 100 messages 101,9 MB each using 10 concurrent threads sending 10 messages each.

#### Consumer test - Scenario 1

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Test ID 1 | Concurrent Threads | Requests Per Thread | Total number of GET calls made | Message Size (MB) | Total Messages Received | Total Message Size Received (GB) | Duration |
| 1 | 1 | 10 000 | 10 000 | 10,9 | 23 000 | 244,82 | 170m 13s |
| 2 | 1 | 10 000 | 10 000 | 10,9 | 23 159 | 246,52 | 170m 34s |

When the test was started the Kafka topic were having 100 000 messages stored. The GET command was receiving in average 2,3 message per call.

### Scenario 2 – Database Event Sourcing with Kafka Consumer



1. Items were loaded to the database using a PLSQL-command. Different number of messages was generated.
2. Kafka Connect gets the records added.
3. Kafka Connect adds the messages to the topic in AVRO format.
4. A Node js consumer app using kafka-avro retrieves the latest offset = item in Avro format.
5. The consumer app loads the Item in the target database.

This scenario focus on distributing data from an Oracle producer database to an Oracle consumer database via the Kafka connect and a Node js application. The reason for testing this is that IKEA is known to have many data replication integrations. It also make use of a 3rd party library to explore how the Kafka connectors works.

All nine tests showed same throughput with the average of consumer to save 13,4 records per second to database. The limitation is within the consuming client. No optimization actions were taken during the test. The producer load to the Kafka instance is considered to be fast – 100 000 records loaded in 11 ms.

#### Producer and Consumer are both up – Scenario 2

This test is to run in an environment thought to be the common running mode scenario. Producer + Kafka environment + Consumer is all up and running.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Record Count | Duration | Rows per second | Producer Record Load Time |
| 1 | 1 000 | 1m 17s | 12,98701299 | 2ms |
| 2 | 10 000 | 12m 34s | 13,26259947 | 2ms |
| 3 | 100 000 | 126m 34s | 13,16829076 | 10ms |

#### Producer down, Consumer up – Scenario 2

This test simulate that the Kafka environment is stopped when the updates are made to the source database.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Record Count | Duration | Rows per second | Producer Record Load Time |
| 1 | 1 000 | 1m 13s | 13,69863014 | 2ms |
| 2 | 10 000 | 12m 20s | 13,51351351 | 2ms |
| 3 | 100 000 | 130m 47s | 12,74372372 | 10ms |

#### Producer up, Consumer down – Scenario 2

The test monitor the behaviour when the target system is not running while producer is updating database and Kafka environment is up running in normal mode.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Record Count | Duration | Rows per second | Producer Record Load Time |
| 1 | 1 000 | 1m 10s | 14,28571429 | 2ms |
| 2 | 10 000 | 12m 22s | 13,47708895 | 2ms |
| 3 | 100 000 | 124m 38s | 13,37255951 | 11ms |

### Additional tested scenarios, infrastructure

Besides the use cases defined above some additional tests was performed to learn and understand how Kafka behaves in an expected IKEA installation.

###### Distributed Kafka instances

Due to the distributed setup of the API management platform with API-gateways installed in various environments like on IKEA premises, AWS, Microsoft Azure, etc. it is necessary to know how distributed Kafka instances behaves and what the best practise says about this type of configuration.



Best practise for implementing Apache Kafka and Zookeeper is to install one Kafka cluster per data centre to achieve lowest possible network latency. The rationale is that Kafka nodes are synchronously updated and can only perform at the speed allowed by the slowest node.

Therefore Apache Kafka comes with “Kafka mirror maker tool” which mirrors a selected sets of Kafka topics from one Kafka cluster to another Kafka cluster.

# Findings

Kafka is available either as open-source license free version or in a licensed version supported by Confluent. The licensed version comes with additional tooling and support. The Kafka ecosystem has got a surrounding community which is very active and supportive. There exists books, webinars, user groups and conferences globally.

Kafka instance(s) will be of middleware type meaning that it will exist between the provider and consumer of data. It can be scaled horizontally.

### Complexity

The overall opinion is that Kafka is considered to be of medium complexity to introduce.

To download and start up **development** in default mode is considered to be simple. Configuration was needed to start sending large message sizes. To find and set the thresholds for values caused some troubles. When integrating system components using Kafka there exists many open source connectors that can be used. There are no really technology that can’t be a Kafka producer or consumer using either native libraries, REST-proxy or other Kafka-connectors. Kafka can be set up in containerized mode enable every developer to run full integration scenarios on local laptop. This can enable local development and contribute to DevOps process/architecture and micro service architecture.

To **operate** the solution is considered to be medium to complex. E.g. cluster configuration, distributed data center replication etc.

### Capacity

When doing database to database scenario the throughput was good. 790 records was sent per minute using off the shelf configuration. This will give 1,15 million of records to be sent per day.

Sending large messages is possible but message size has got impact on throughput. Adding Kafka to the IKEA IT-landscape will not remove the need of system design when working with interfaces for publishing data. In one test scenario 100 000 messages each of 10,9 MB resulting in more than 1 TB of data was sent during 103 minutes.

The test scenarios 1 and 2 was run in a single Virtual Machine and using Docker Containers for different components of the Kafka infrastructure. Higher throughput can be achieved by implementing clustering configuration and further fine tuning the infrastructure. Kafka stores all messages in the file system, meaning that careful considerations have to be made in terms of storage (e.g. file system used, type of storage) as the performance of storage will be directly proportional to the performance of the Kafka infrastructure.

There exists many benchmarks on throughput made by different organisations on the internet.

### Functionality

Kafka can be used in both as ETL/batch and Pub/Sub use cases. It enables data to be enriched both on inbound or outbound flow depending on requirements. Kafka supports the VETRO pattern. Transformation can be done both during inbound flows as well as on outbound flows.

### Operational aspects

Support of Kafka infrastructure could be handled by existing middleware infrastructure team within IKEA. Training must be provided to the supporting organisation.

# Recommendation

The recommendation is to evaluate Kafka in a real use case. Kafka has been proven to be used for ETL and Pub/Sub scenarios.

Using Kafka will still need provider and consumer to define the information model. You can’t just dump your data using Kafka. Governance support is needed and should be integrated with running components.

Explore how Kafka can be used for Business Intelligence purpose using e.g. KSQL in enrichment of data, how databases can be used as source sending near real time events.

Explore how Kafka need to be set up when IKEA systems are running in different data centers operated by different vendors.

Explore different topologies such as no container virtualization or the recently announced Kubernetes Operator.

Explore supporting tools for Kafka. Monitoring – operational, monitoring – business transactions, governance – system-to-system dependencies

*May 2018 Harri Willstrand, Niklas Bergkvist, Luis Weir, James Neate*