# MEP 28 -- Support Milvus Kafka

## Motivation

The log broker is a pub-sub system within Milvus, It is responsible for streaming data persistence,  event notification, recovery etc.  Now Milvus cluster mode uses Pulsar as a log broker, and standalone mode uses RocksDB.

Apache Kafka is a distributed event store and stream-processing platform, and it is a popular solution for data streaming needs.  Many community users expect Milvus to support Kafka because they have already used it in the production environment.
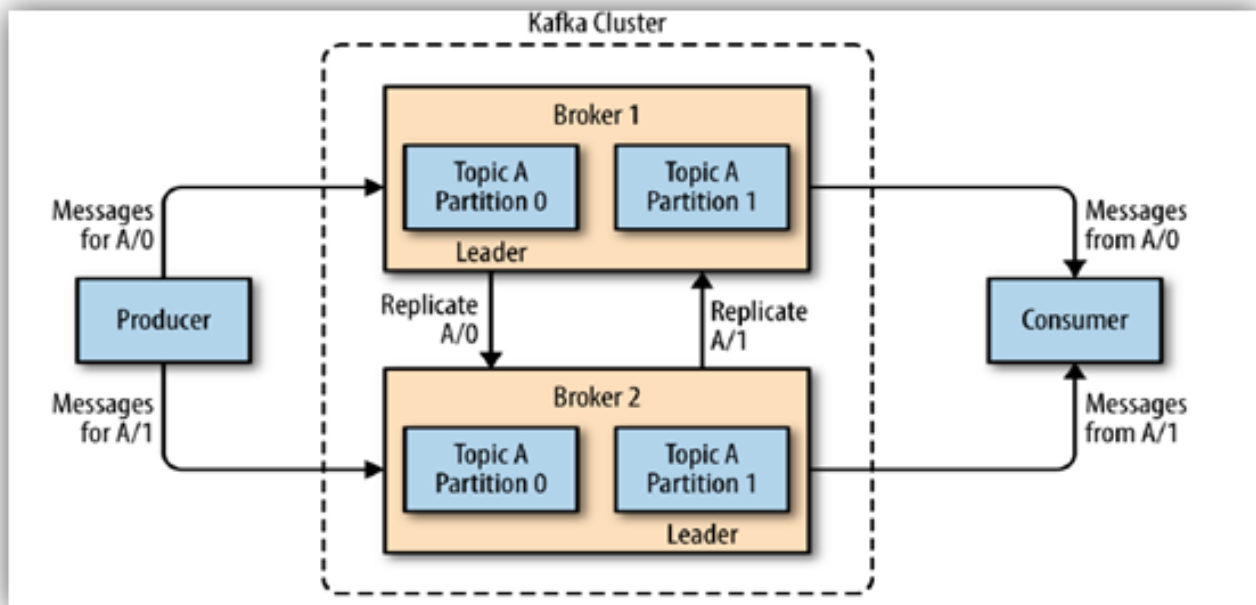


Image Source

## Summary

 Milvus supports Kafka as a message stream,  we can use the configuration option to decide to use Pulsar or Kafka on cluster mode.

**milvus.yaml**

```
kafka:
  brokers:
        - localhost:9092
```

## Design Details

**Configuration**

If we want to enable Kafka, need to config it in milvus.yaml file, Pulsar is also.  when Kafka and Pulsar config at the same time, Pulsar has higher priority use.

**Kafka Client SDK**

We tried using sarama and confluent-kafka-go in our development and found that there was basically no difference in the producer. But there is a big difference when using consumer groups.

Sarama uses consumer group needs to implement Sarama interfaces. it is very difficult to control and hard to seek.

confluent-kafka-go use consumer group to consume messages just a function. It is very simple to use. This function allows you to directly set the offset from which to start consumption.

**Interface Implementation**

Note:  in order to provide a unified MQ interface,  We will remove a series of Reader API from MsgSream interface, actually Kafka does not support Reader API and it also is implemented by Consumer API.

Implement Kafka consumer based on the following interface:

**kafka_consumer.go**

```
// Consumer is the interface that provides operations of a consumer
type Consumer interface {
        // returns the subscription for the consumer
        Subscription() string

        // Message channel
        Chan() <-chan Message

        // Seek to the uniqueID position
        Seek(MessageID, bool) error //nolint:govet

        // Ack make sure that msg is received
        Ack(Message)

        // Close consumer
        Close()

        // GetLatestMsgID return the latest message ID
        GetLatestMsgID() (MessageID, error)
}
```

Implement Kafka producer based on the following interface:

**kafka_producer.go**

```
// Producer is the interface that provides operations of producer
type Producer interface {
        // return the topic which producer is publishing to
        //Topic() string

        // publish a message
        Send(ctx context.Context, message *ProducerMessage) (MessageID, error)

        Close()
}
```

on the other hand,  we must provide a configurable ability for MQ clients, so  SetParams API will be replaced by Init API.

**mq_factory.go**

```go
type Factory interface {
        #SetParams(params map[string]interface{}) error
        Init(params *paramtable.ComponentParam) error
}
```

**Deployments**

- standalone
    - docker

**dev/docker-compose.yml**

```yaml
version: '3.5'

services:
  zookeeper:
    image: 'bitnami/zookeeper:3.6.3'
    ports:
      - '2181:2181'
    environment:
      - ALLOW_ANONYMOUS_LOGIN=yes
  kafka:
    image: 'bitnami/kafka:3.1.0'
    ports:
      - '9092:9092'
    environment:
      - KAFKA_BROKER_ID=0
      - KAFKA_CFG_LISTENERS=PLAINTEXT://:9092
      - KAFKA_CFG_ADVERTISED_LISTENERS=PLAINTEXT://127.0.0.1:9092
      - KAFKA_CFG_ZOOKEEPER_CONNECT=zookeeper:2181
      - ALLOW_PLAINTEXT_LISTENER=yes
      - KAFKA_CFG_MAX_PARTITION_FETCH_BYTES=5242880
      - KAFKA_CFG_MAX_REQUEST_SIZE=5242880
      - KAFKA_CFG_MESSAGE_MAX_BYTES=5242880
      - KAFKA_CFG_REPLICA_FETCH_MAX_BYTES=5242880
      - KAFKA_CFG_FETCH_MESSAGE_MAX_BYTES=5242880
    depends_on:
      - zookeeper

networks:
  default:
    name: milvus_dev
```

- Cluster
    - Helm Chart
    - Operator

# Test Plan

- pass the ut
- performance testing
- chaos testing