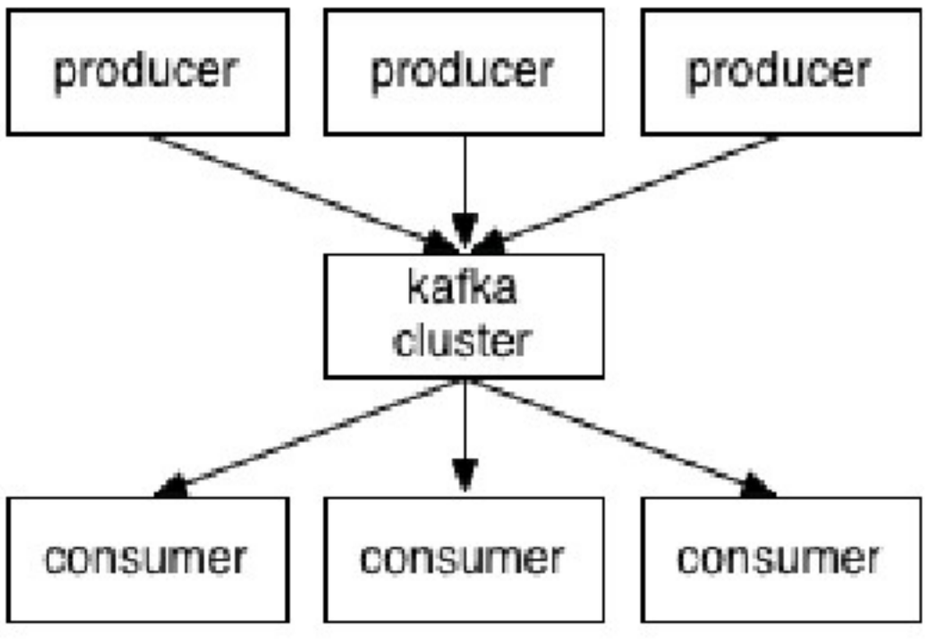
Kafka简介

Kafka是由LinkedIn开发分布式消息系统，使用scala编写，因可水平扩展和高吞吐量而广泛使用。
Cloudera、Storm、spark都可以与之集成。

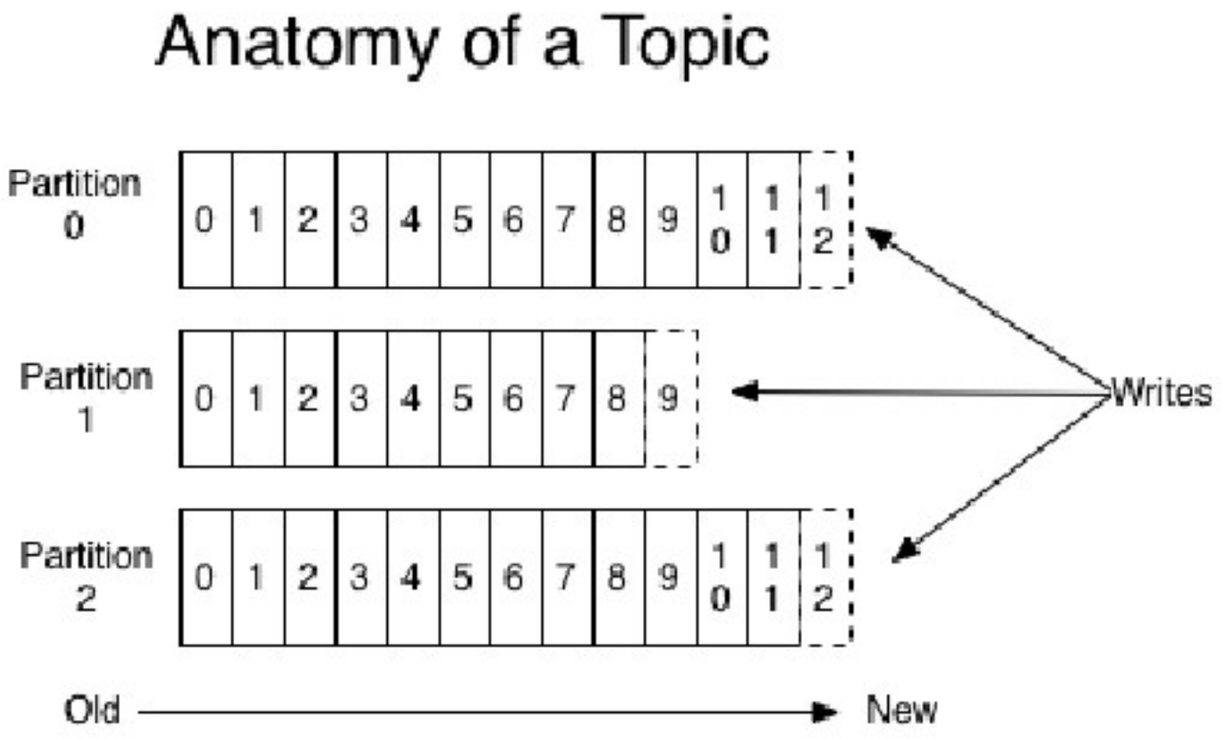
生产者通过网络发送消息给kafka集群，由消费者消费。



Kafka简介

- Topic

  消息被发送时依据的目录。每个主题在集群中都维护了分区的log。



Kafka简介

- Topic

  每个分区都是有序、不可变的消息序列，持续追加在提交log末端。分区中的消息都指定了序列号称为offset，唯一标识分区中的消息。

  kafka集群保留所有已发布消息，不论是否消费，可以通过配置时间周期进行控制。

  consumer中唯一保留的元数据就是位置，称为offset。offset由consumer控制，通常consumer在读取消息时早于offset，但实际上可以任何顺序处理消息。

Kafka简介

- Topic

  consumer来去自由，对别人没有任何影响。

  分区机制使得log超过单个server的容量，单纯一个分区必须符合所在server的容量，但往往一个主题可以有多个分区。

  分区用于并行单元的处理。

Kafka简介

- 分布式

  log的分区被分发到集群的server上，每个server都处理数据并请求分区的共享。每个分区在集群上配置副本用于容错。

每个分区都有一个server作为leader，其他作为follower。leade处理分区的所有读写请求，follower被动复制leader。

Kafka简介

- Producer

  发布消息到主题，负责消息指派给主题中的哪个分区。可以使用轮询方式或key方式。可以同步也可以异步，producer还有buffer，可以缓冲数据，批量发送。

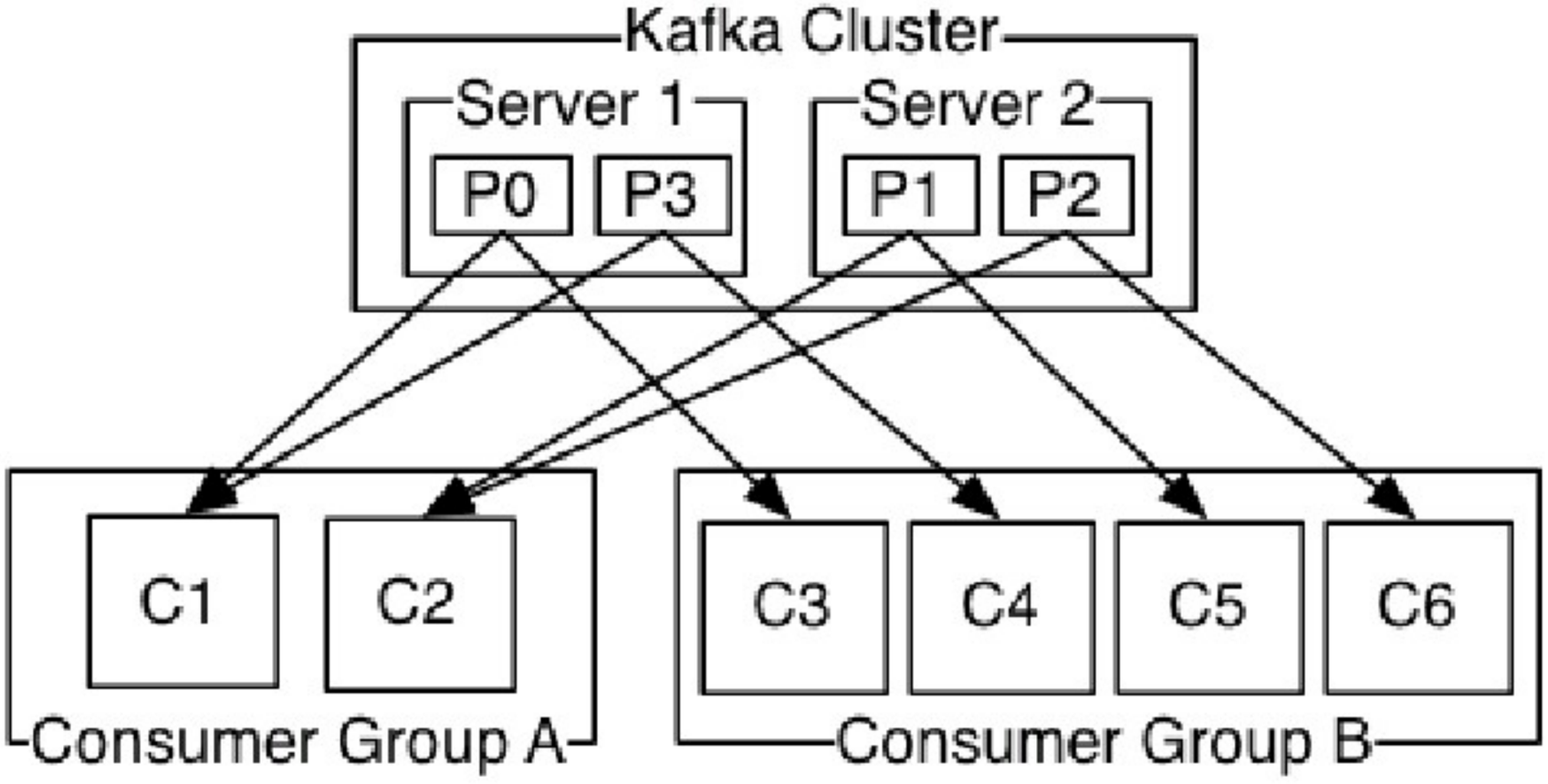
- Consumer

  队列模式下，消息被一个consumer消费，发布订阅模式消息被多个consumer消费。kafka做了抽象，定义了consumer group，每个group中只有一个consumer能够消费topic上的消息。将队列模式和主题模式统一起来了。

Kafka简介

- Consumer

  相较于传统消息系统，kafka有着强顺序性。

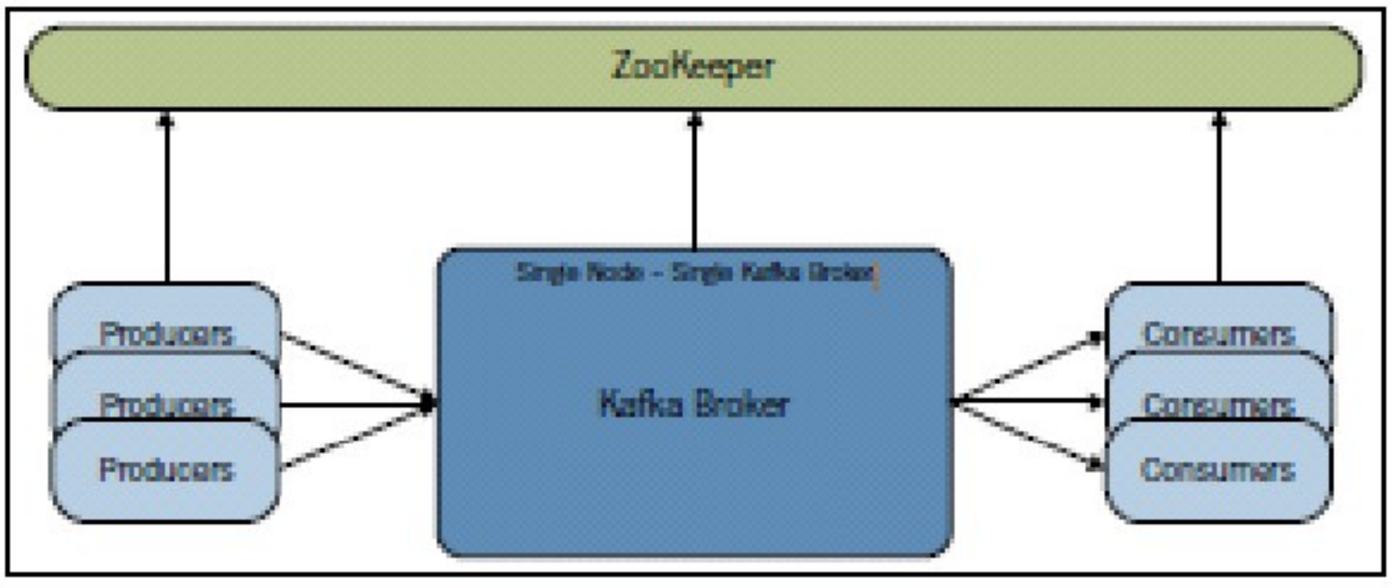


安装Kafka

- 下载kafka
  kafka_2.11-0.9.0.0.tgz
- 解压Kafka
  $>tar -xzvf kafka_2.11-0.9.0.0.tgz
- 安装所需项
  $>jdk1.6+

安装Kafka集群

- Single Node-Single Broker
  在一台主机上安装kafka集群。



安装Kafka集群

- Single Node-Single Broker
  - 启动zk服务器
    $>bin/zookeeper-server-start.sh config/
    zookeeper.properties
    [zk.properties]
    dataDir=/tmp/zookeeper
    clientPort=2181
  - 启动broker
    $>bin/kafka-server-start.sh config/server.properites
    [server.properties]
    Broker.id=0
    log.dir=
    zookeeper.connect=

安装Kafka集群

- Single Node-Single Broker
  - 启动zk服务器
    $>bin/zookeeper-server-start.sh config/
    zookeeper.properties
    [zk.properties]
    dataDir=/tmp/zookeeper
    clientPort=2181
  - 启动broker
    $>bin/kafka-server-start.sh config/server.properites
    [server.properties]
    Broker.id=0 | log.dir= | zookeeper.connect=

安装Kafka集群

- Single Node-Single Broker
  - 创建Kafka主题

```
$>bin/kafka-create-topic.sh --zookeeper locahost:2181
                            --replica 1
                            --partition 1
                            --topic kakfatopic
```

  - 启动生产者发送消息

```
$>bin/kakfa-console-producer.sh
                        --broker-list localhost:9092
                        --topic kafkatopic
```

安装Kafka集群

- Single Node-Single Broker
  - 启动Consumer消费消息

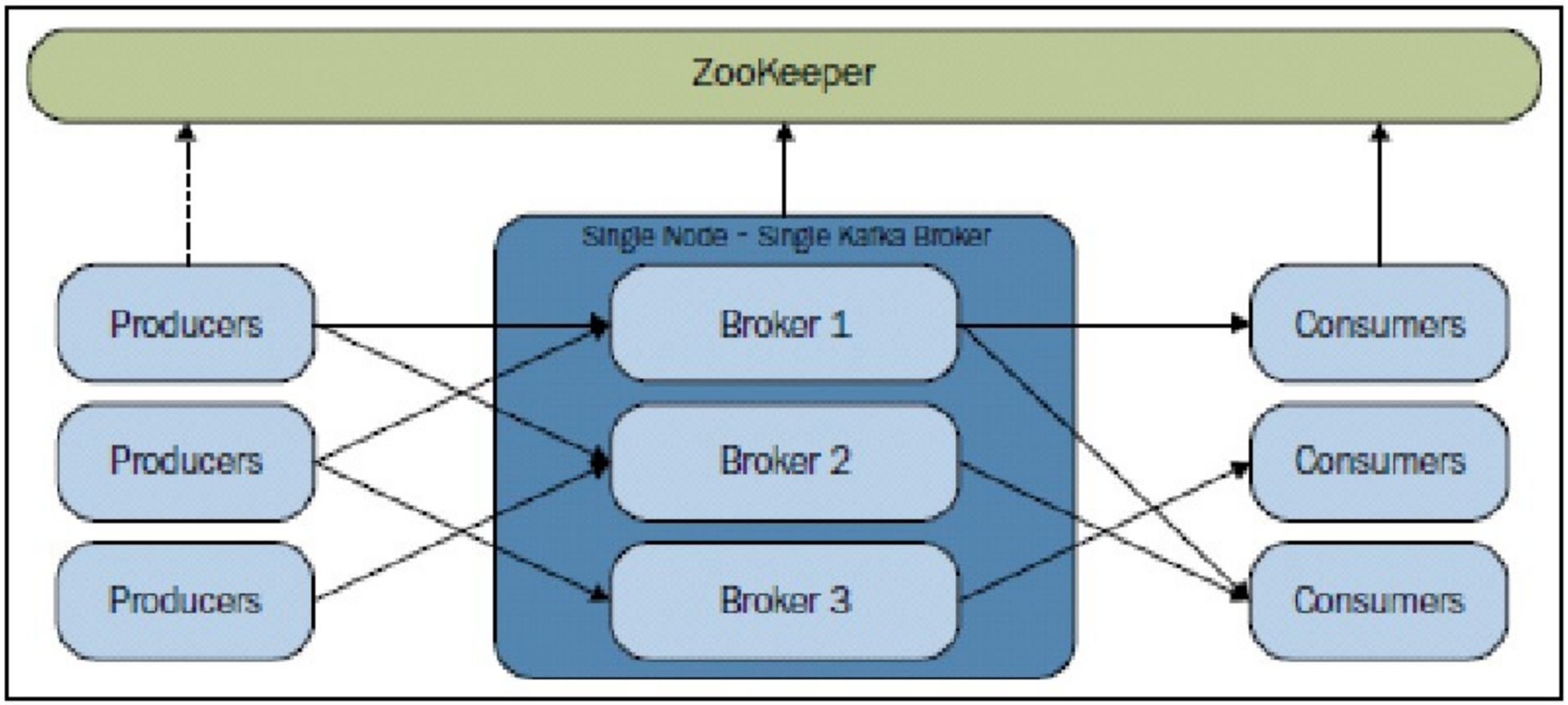
```
$>bin/kakfa-console-consumer.sh
                    --zookeeper localhost:2181
                    --topic kafkatopic
                    --from-beginning
```

Consumer默认配置文件是consumer.properties。

```
groupid=test-consumer-grouper
zookeeper.connect=localhost:2181
```

安装Kafka集群

- Single Node-Multiple Broker
  在单个节点上安装多个broker。



安装Kafka集群

- Single Node-Multiple Broker
  - 启动zookeeper
    同上
  - 启动多个broker，每个对应一个配置文件
    [server-1.properties for broker1]
    broker.id=1
    port=9092
    log.dir=/xx/x/x
    [server-1.properties for broker1]
    broker.id=2
    port=9093
    log.dir=/xx/x/x

安装Kafka集群

- Single Node-Multiple Broker
  - 启动多个broker，每个对应一个配置文件

    $>bin/kafka-server-start.sh server-1.properties

    $>bin/kafka-server-start.sh server-2.properties
  - 创建kafka主题

    $>bin/kafka-create-topic.sh --zookeeper localhost:2181
    --replica 2
    --partition 2
    --topic othertopic

安装Kafka集群

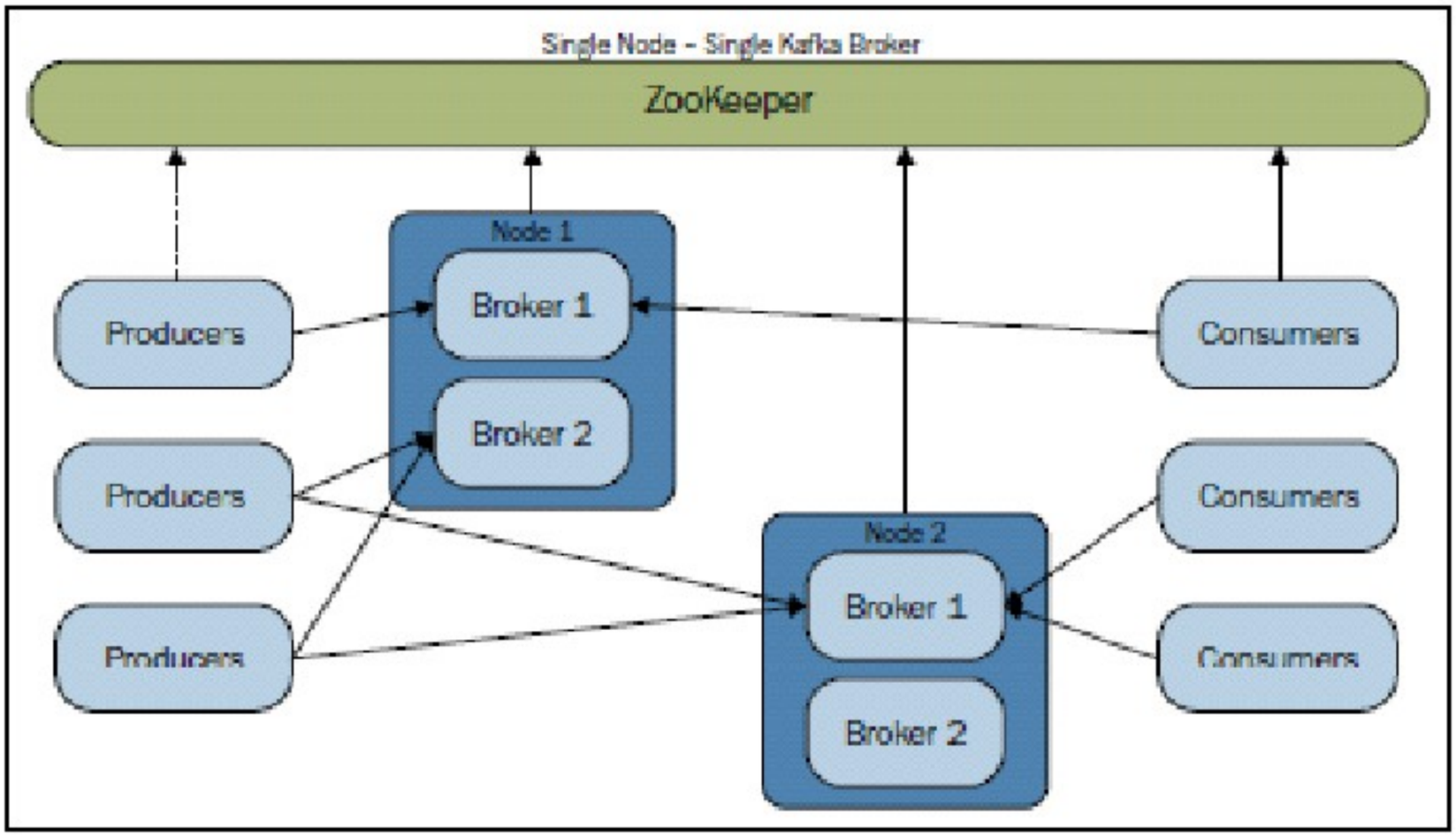
- Single Node-Multiple Broker
  - 创建Producer发送消息

    $>kafka-console-producer.sh
    --broker-list localhost:9092,localhost:9093
    --topic othertopic
  - 启动consumer消费消息

    $>kafka-console-consumer.sh
    --zookeeper localhost:2181
    --topic othertopic
    --from-beginning

安装Kafka集群

- Multiple Node-Multiple Broke
  在多个节点上配置多个Broker组件。



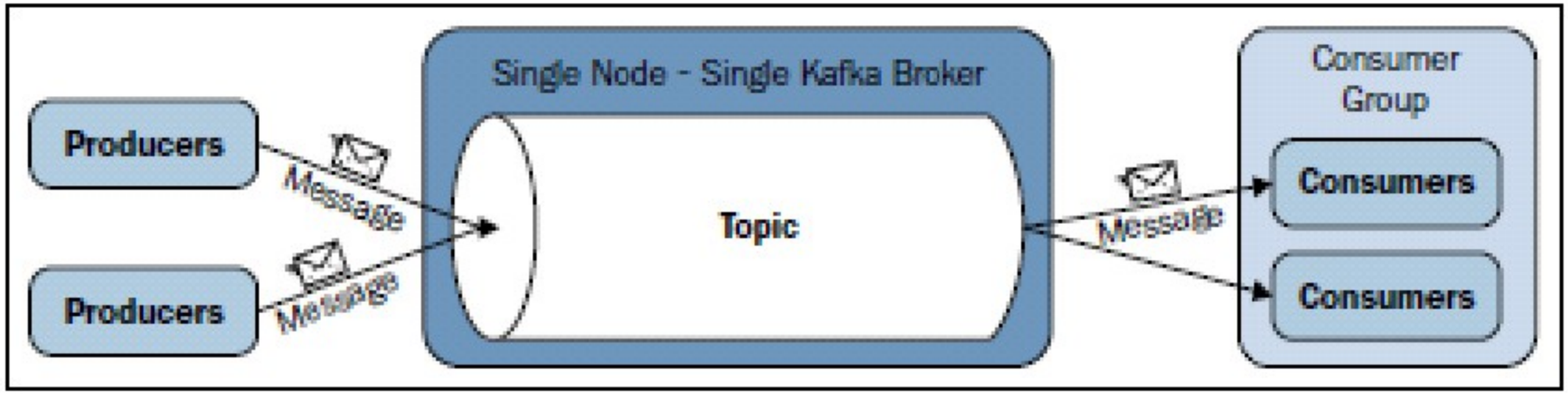
Kafka的设计

- Kafka设计基础
- Kafka中的消息压缩
- kafka的集群镜像
- Kafka的复制

Kafka的设计

- **Kafka设计基础**

  生产者发布消息给主题，主题由broker创建，扮演服务器的角色。消费者订阅主题获得消息。



Kafka的设计

- **Kafka设计基础**

  消费者分成组，每个组中只有一个消费者可以消费消息。消息的状态维护在消费者中，broker没有任何消费记录。因此如果自定义consumer的话，过于简单会导致多次消息处理。

  - kafka的核心是消息缓存和存储。在kafka中，消息被即可写入到OS的内核page。数据的缓存和清理操作是可配置的。
  - 即使消息被处理后也可以驻留更长时间以备重复消费。
  - Kafka对分组消息使用消息集合来允许更少的网络消耗。
  - Kafka中消费者存储在zk中，也可以是其他的OLTP中。
  - kafka中的生产消费以推拉模型工作。

Kafka的设计

- **Kafka消息压缩**
  生产者使用GZIP或snappy压缩消息，消费者解压消息。这会降低网络负载。

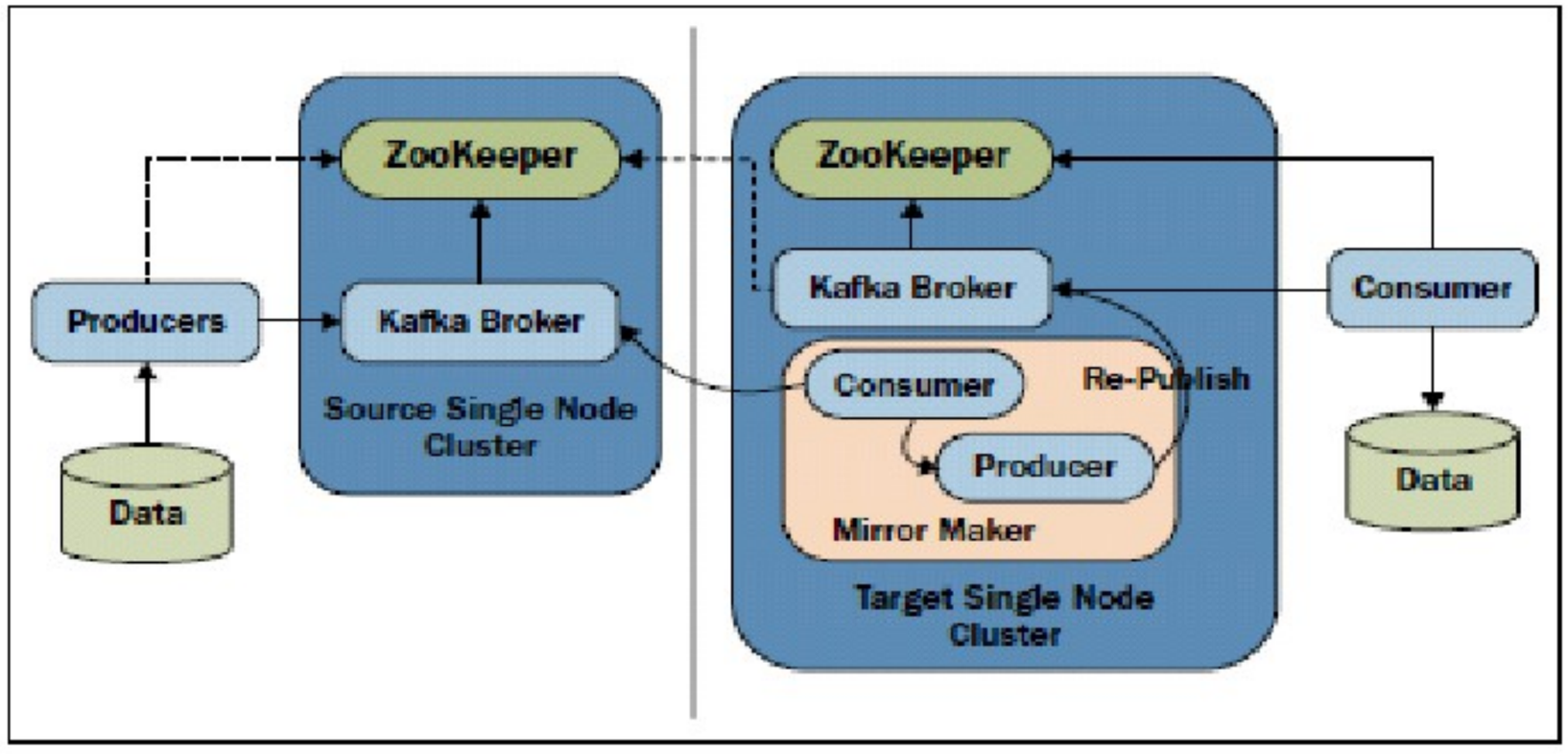
为了在压缩与解压缩消息间进行区分，在消息头信息中引入了压缩属性字节。该字节最低的两个位代表压缩编码器，如果是0表示未压缩的消息。

消息压缩技术在跨数据中心的镜像数据非常有用，届时大量的消息会使用压缩格式进行传输。

Kafka的设计

- **Kafka集群镜像**

  Kafka镜像特性用来创建现有集群的副本。Kafka提供了镜像maker工具。镜像工具的作用是从源集群中消费消息并在目标集群中重新发布消息。



Kafka的设计

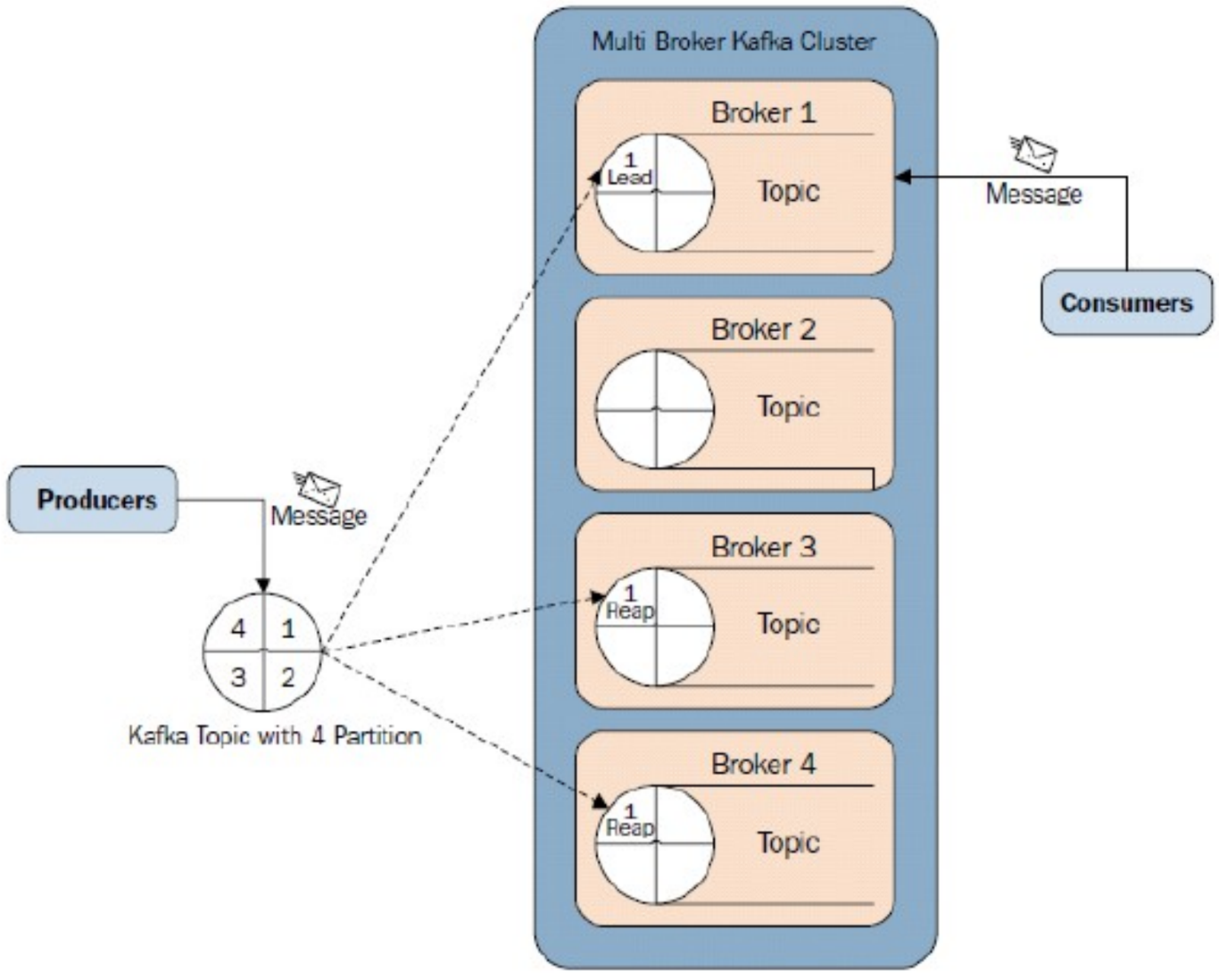
- **Kafka副本**

  Kafka中的消息分区是在Broker中进行的。消息生产者决定消息如何进行分区，broker以相同顺序存储这些消息。分区数量可以在broker中对每个topic尽心配置。

  副本确保了在broker故障的时候，仍能够发布和消费消息。生产者和消费这都是副本化的。

Kafka的设计

- **Kafka副本。**



Kafka的设计

- **Kafka副本**。
  在副本模式下，每个消息分区都有n个副本，可以承受n-1故障来继续消息的分发。有一个副本扮演着lead角色。zk中存放着lead的信息。

每个部分使用本地log和offset存放对应的消息部分并周期性的同步到disk。该过程确保了消息写入到所有副本。

选择新lead的过程就是所有follow注册副本的过程，第一个注册的follow就成为lead。

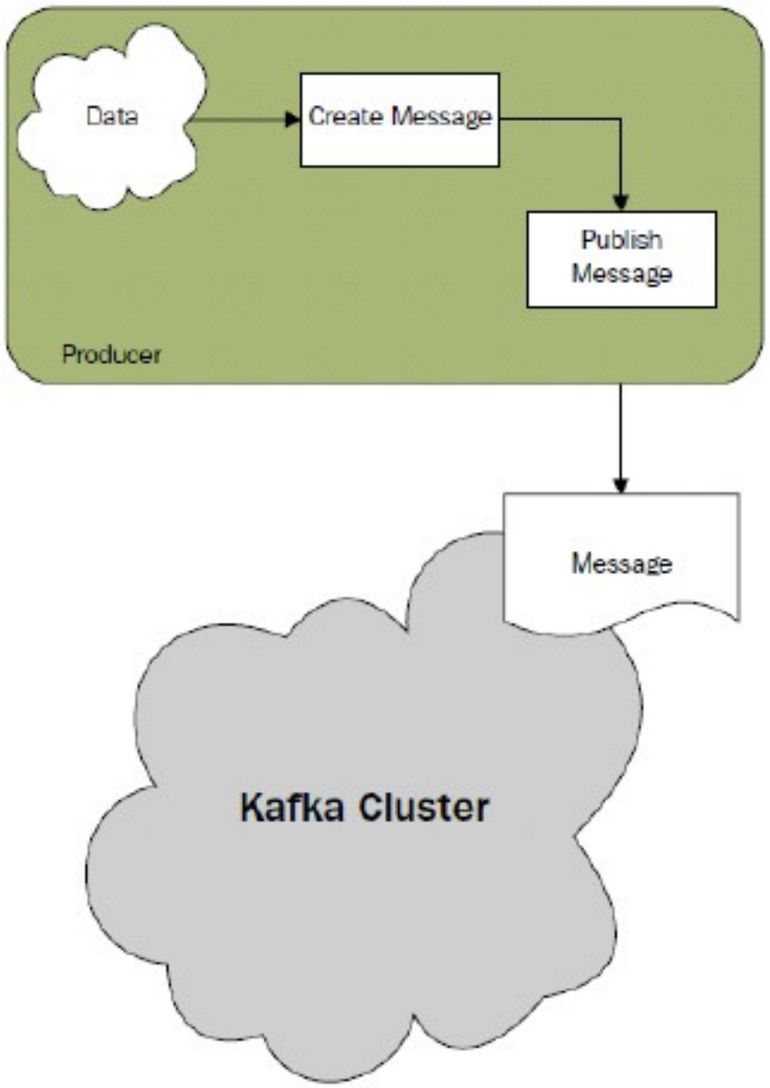
Kafka的设计

- **Kafka副本**。
  Kafka支持两种副本模式
  - 同步复制
    生产者从zk中找到lead然后发布消息，消息发布后，就写入lead的log中然后所有follow使用一个channel都开始pull消息，确保其顺序的正确性。一旦消息写入到相应的log，follow就向lead发送确认信息。当lead收到所有follow的确认信息后，lead再发送确认信息给生产者。
  - 异步复制
    lead将消息写入到log后，立即发送确认信息给客户端，不需要等待follow的确认过程。该模式无法保证在broker故障时消息的分发。

编写生产者

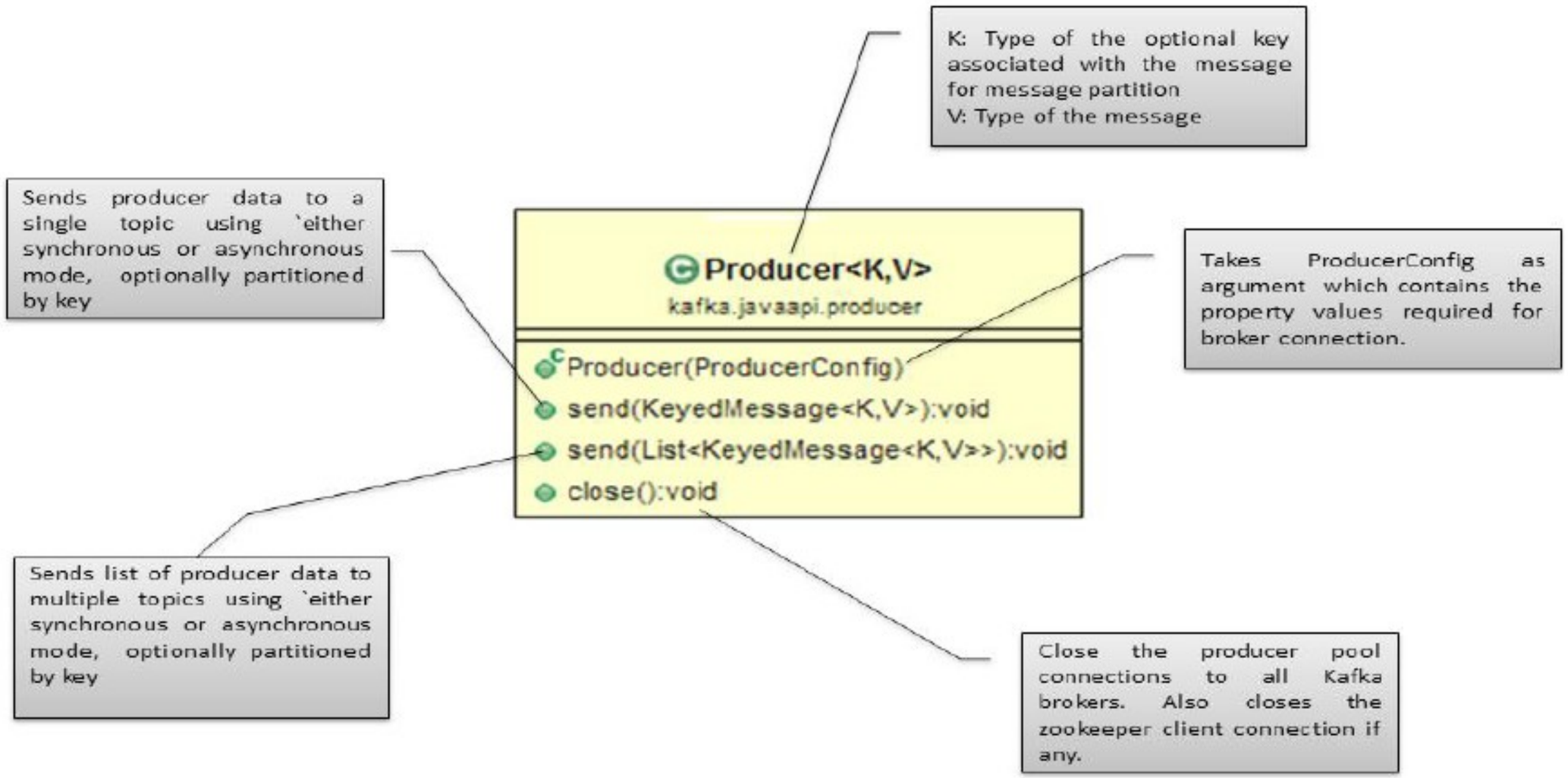
生产者是产生消息并发送到broker，供consumer消费。

- Kafka API
- 基于java的简单 Kafka producer
- 使用消息分区的基于java的Kafka producer



编写生产者

- **java Kafka API**



编写生产者

- **java Kafka API**

  Producer是泛型化类，k是分区的类型，v是分区的值。

  //主题、key、value

  class KeyedMessage[K,V](val top:String,val key:K,val m:V)

  //封装诸如BrokerList、分区类、串行化类等信息

  class ProducerConfig

编写生产者

java Kafka API

构建消息并进行发送

```
props.put("broker.list", "localhost:9092");
props.put("serializer.class", "kafka.serializer.StringEncoder");
props.put("request.required.acks", "1");
producer = new Producer<Integer, String>(new ProducerConfig(props));

String messageStr = new String("Hello from Java Producer");
KeyedMessage<Integer, String> data
        = new KeyedMessage<Integer,String>(topic, messageStr);
producer.send(data);
```

完整代码见备注。

编写生产者

消息分区的Producer

```
props.put("partitioner.class", "test.kafka.SimplePartitioner");
props.put("request.required.acks", "1");
Producer p = new Producer<Integer,String>(config);
```

```
package test.kafka;
import kafka.producer.Partitioner;
public class SimplePartitioner implements Partitioner<Integer> {
    public int partition(Integer key, int numPartitions) {
        return key % numPartitions;
    }
}
```

完整代码见备注。