**Content:**

**Internals of Kafka Producer**

**Offset**

**Internals when producer sends message**

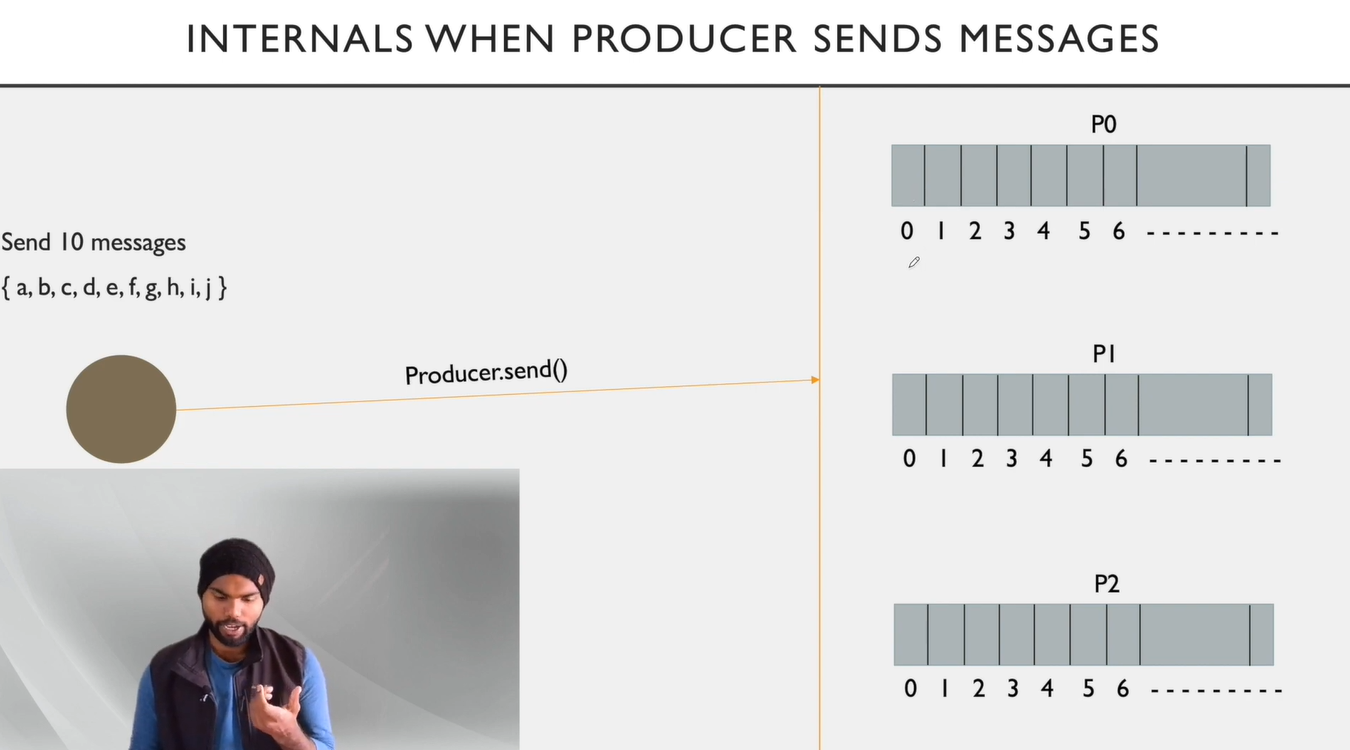


Figure 1 When producer sends messages

* When producer sends any message, it calls an API (producer.send()).
* Producer can send message on topic level and partition level.
* Whenever producer sends any messages to topic **by default** it will send messages in round robin fashion.

**Offset:**

* Whenever a message gets stored in a partition it gets a sequence\_id called *offset*.
* The records in the partitions are each assigned a sequential id number called the *offset* that uniquely identifies each record within the partition.
* Offset starts from 0.
* Only partitions have offsets.
* Topics don’t have offsets.

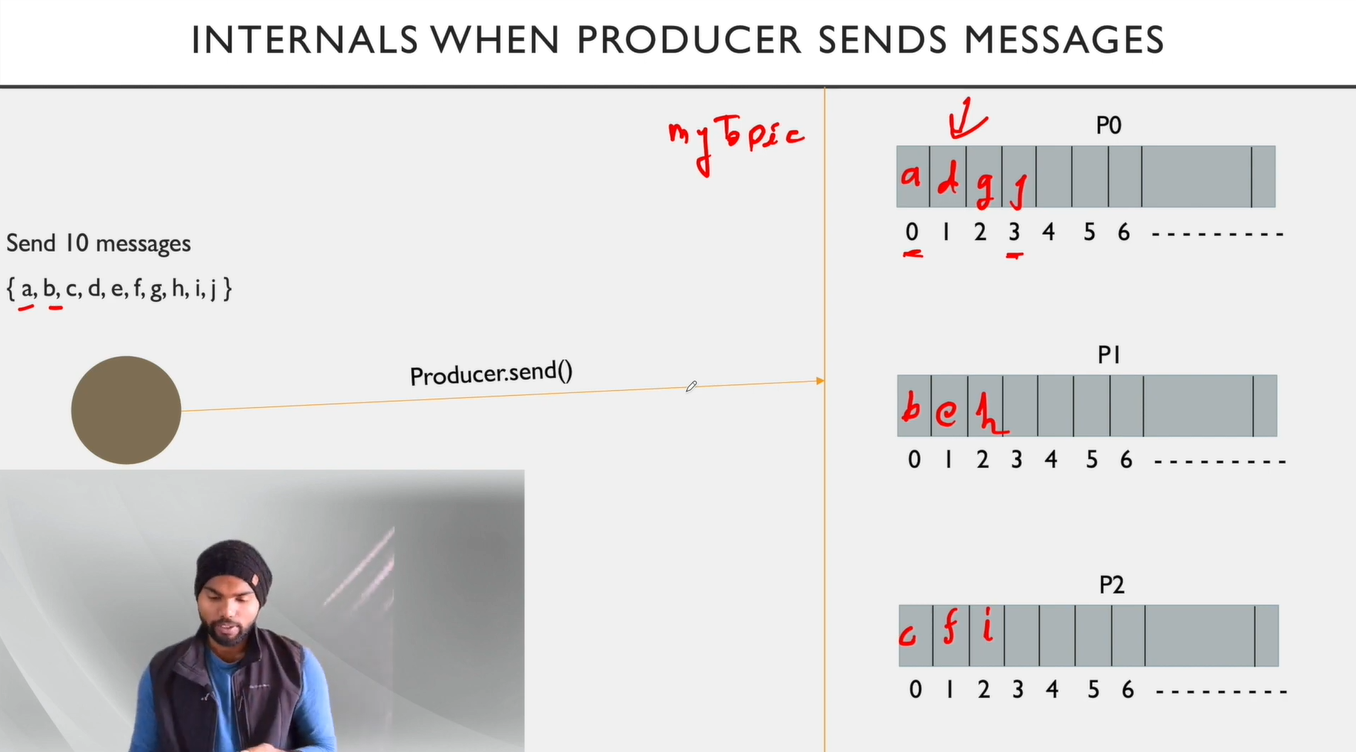


Figure 2 Offsets

* In above image we can see we sent 10 messages from a to j.
* Each message got sent in round robin fashion in each partition.
* Each message has its own unique sequence id. (e.g. a=0, d=1)
* There are three types of offset:

1. **Log-end offset:**

Offset of the last message written to a log/Partition.

In Our example P0 has log-end offset 3. And P1 and P2 has log-end offset 2.

1. **Current offset:**

Pointer to the last record that Kafka has already sent to a consumer in the most recent poll.

Kafka cluster stores information of how many messages are sent to consumer groups in ***current offset***.

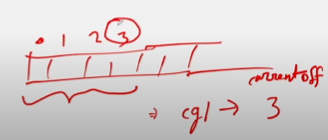


Figure 3 Current offset

In above example Kafka cluster sent 4 messages to consumer group and current offset is 3 as the last index of sent message is 3.

When Kafka cluster sends messages to **consumer group** it keeps record of this, using ***current offset***.

It is closely related to consumer group.

1. **Committed offset:**

Marking an offset as consumed is called committing an offset(Committed offset).

When consumer receives message from kafka cluster and complete the processing or consume the message it sends an acknowledgement to Kafka cluster that it has completed the processing of message, it is called committed offset.

It is **not possible** that committed offset can be more than current offset. It can be less than or equal to current offset. Because first consumer will receive message from producer and then only it can commit it.

It is closely related to consumer group.

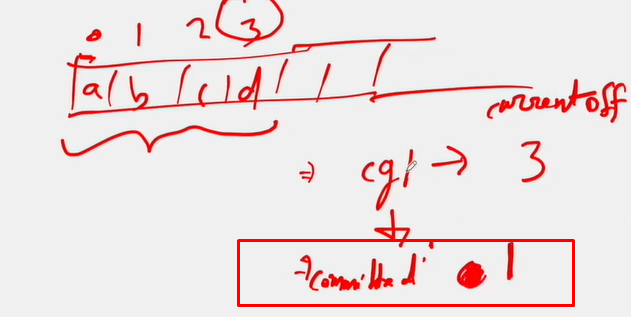


Figure 4 Committed offset

In above example when consumer is done with processing of message ‘b’ Kafka cluster will save this information in committed offset that is with index number ‘2’.

When consumer is giving the acknowledgement to kafka cluster that message is consumed, for this process there are two types:

1. Auto commit
2. Manual commit
   1. All messages in one go
   2. One message at one time
3. **Auto Commit:**

It automatically sends information to broker that messages are consumed after particular time interval.

But there is one drawback that if sometime any message is crashed, or consumer didn’t processed the message in given time it will auto commit the message and there will be data loss.

1. **Manual Commit:**

In manual commit consumer tells cluster that It has consumed the messages then only cluster will update committed offset.

There are two types of manual commit:

1. All messages in one go:

In this type consumer sends acknowledgement of committed messages at one time.

For example, consumer receives 15 messages it will process on all the messages and at last it will send the acknowledgement to cluster.

1. One message at one time:

In this type consumer sends acknowledgement of processed message immediately once the message is consumed or processed successfully.

For example, if we have 15 messages from ‘a to o’ when message a is processed immediately consumer will send acknowledgement to cluster.

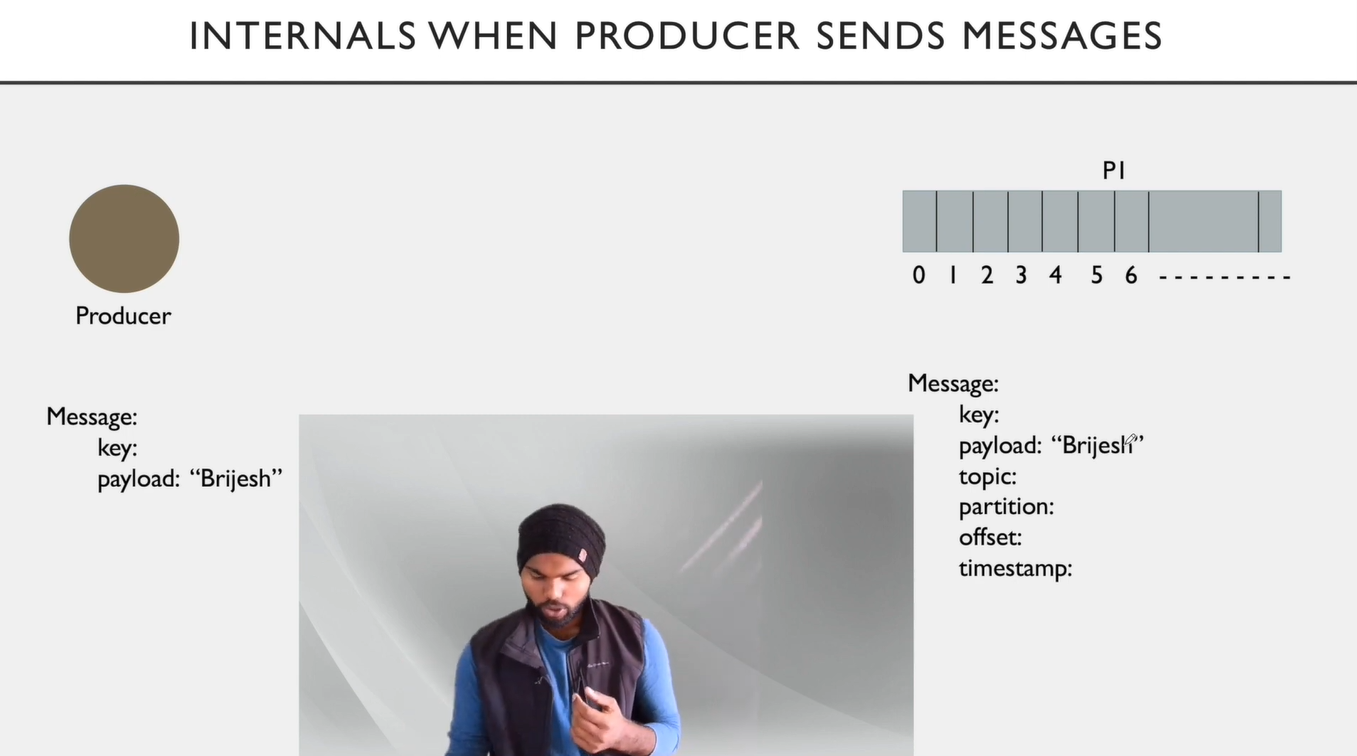
* **Properties getting sent when producer sends messages:**

Figure 5 Properties when producer sends message

* When producer sends a message, it sends two things **key** and **payload**.
* Key is used when we are sending messages on partition level.
* When we are sending messages using ***key***, producer will send messages to the mentioned key. Else messages will go in round robin fashion to each partition.
* If we haven’t mentioned key it will be ***null*** by default.
* When Kafka is storing messages in file system it stores **metadata** with it.
* In below example we can see which information a broker stores when a message is getting stored in a file system:

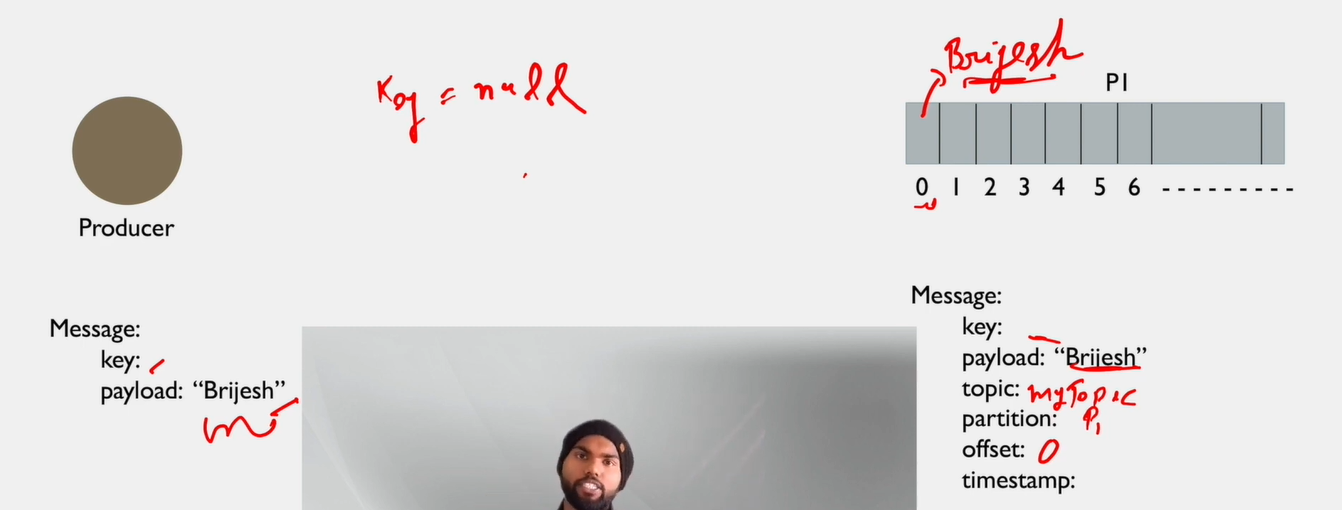


Figure 6 Message metadata or information

* In above example timestamp is when broker saves message in its file system.