[Kafka Producers | Learn Apache Kafka with Conduktor](https://www.conduktor.io/kafka/kafka-producers)

Graphical user interface, text

Description automatically generated

So we have load balancing in this case

because your producers, they're going to send data

across all partitions based on some mechanism,

and this is why Kafka scales,

it's because we have many partitions within a topic

and each partition is going to receive messages

from one or more producers.

So now producers have message keys in the message.

So the message itself contain data,

but then we can add a key and it's optional,

and the key can be anything you want,

could be a string, a number, a binary, et cetera, et cetera.

So you have two cases. So if the key is null,

then the data is going to be sent round robin.

So that means that it's going to be sent to partition zero

then partition one, then partition two and so on,

and this is how we get load balancing, okay?

Key equals null means that the key was not provided

in the producer message,

but if the key is not null,

that means that the key has some value.

It could be again, a string, a number, a binary,

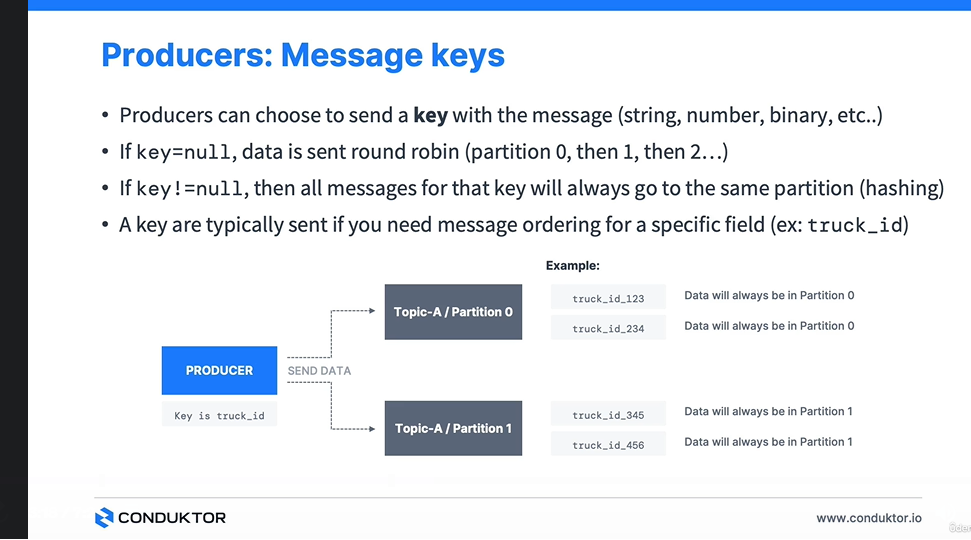
whatever you want.

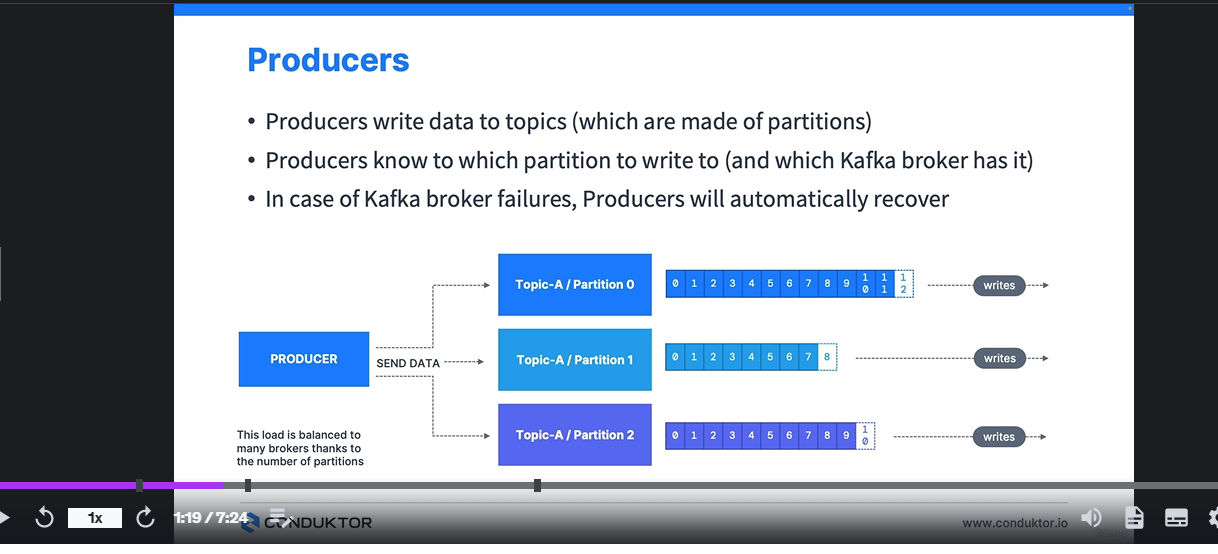
And the Kafka producers have a very important property,

that is that all the messages that share the same key

will always end up being written to the same partition,

thanks to a hashing strategy,





In many programming languages, the key and value are represented as objects, which greatly increases the code readability. However, Kafka brokers expect byte arrays as keys and values of messages. The process of transforming the producer's programmatic representation of the object to binary is **called message serialization**.



A Kafka partitioner is a code logic that takes a record and determines to which partition to send it into.

This is just to stress the fact that

producers are the one who choose

where the message is going to end up,

thanks to the key bytes, okay?

By hashing the key.

**Consumer groups**

for horizontal scalability purposes it is recommended to consume Kafka topics as a group.

Consumers that are part of the same application and therefore performing the same "logical job" can be grouped together as a Kafka consumer group.

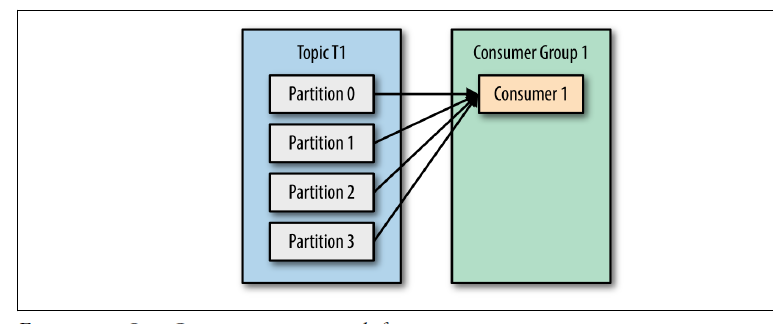
A topic usually consists of many partitions. These partitions are a unit of parallelism for Kafka consumers.

The benefit of leveraging a Kafka consumer group is that the consumers within the group will coordinate to split the work of reading from different partitions.

CONSUMER-PARTITION MAPPING

It is important to note that each topic partition is only assigned to one consumer within a consumer group, but a consumer from a consumer group can be assigned multiple partitions.

Many to one is possible. One partition to many consumers not possible.



Usually, we have as many consumers in a consumer group as the number of partitions. If we want more consumers for higher throughput, we should create more partitions while creating the topic. Otherwise, some of the consumers may remain inactive.

 the consumer will regularly **commit** the latest processed message, also known as **consumer offset**.

 and the responsible Kafka broker will ensure writing to the \_\_consumer\_offsets topic

The process of committing offsets is not done for every message consumed (because this would be inefficient), and instead is a periodic process.

Offsets are critical for many applications. If a Kafka client crashes, a rebalance occurs and the latest committed offset help the remaining Kafka consumers know where to restart reading and processing messages.

In case a new consumer is added to a group, another consumer group rebalance happens and consumer offsets are yet again leveraged to notify consumers where to start reading data from.

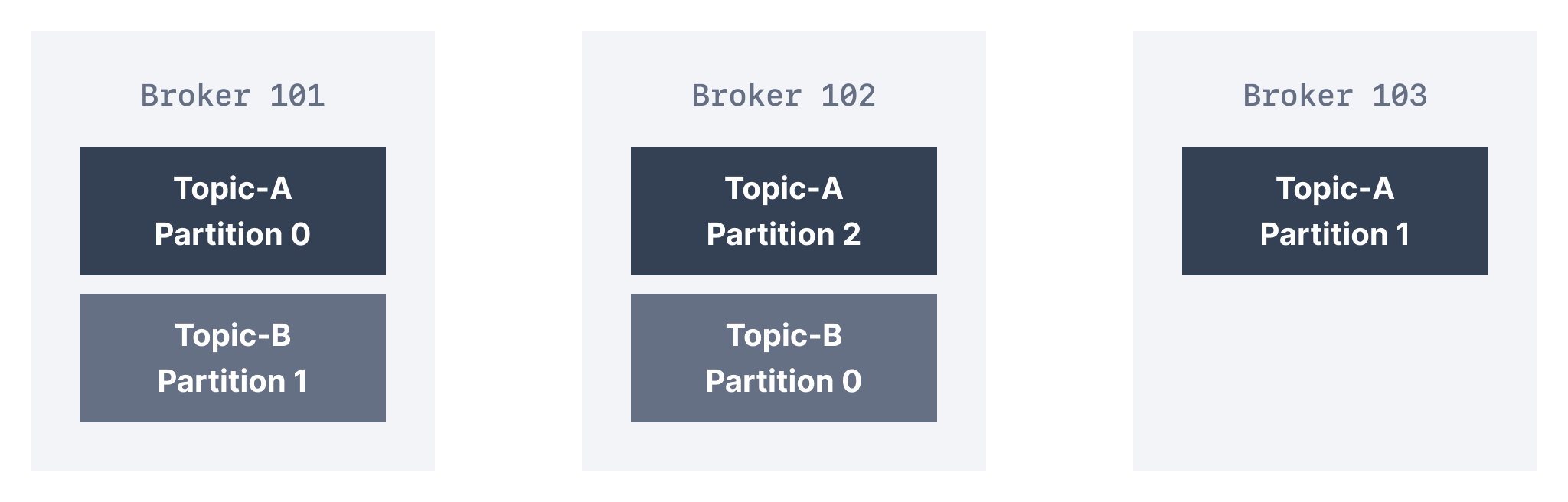
Therefore consumer offsets must be committed regularly.

A single Kafka server is called **a Kafka Broker**. That Kafka broker is a program that runs on the Java Virtual Machine (Java version 11+) and usually a server that is meant to be a Kafka broker will solely run the necessary program and nothing else.

An ensemble of Kafka brokers working together is called a **Kafka cluster.**

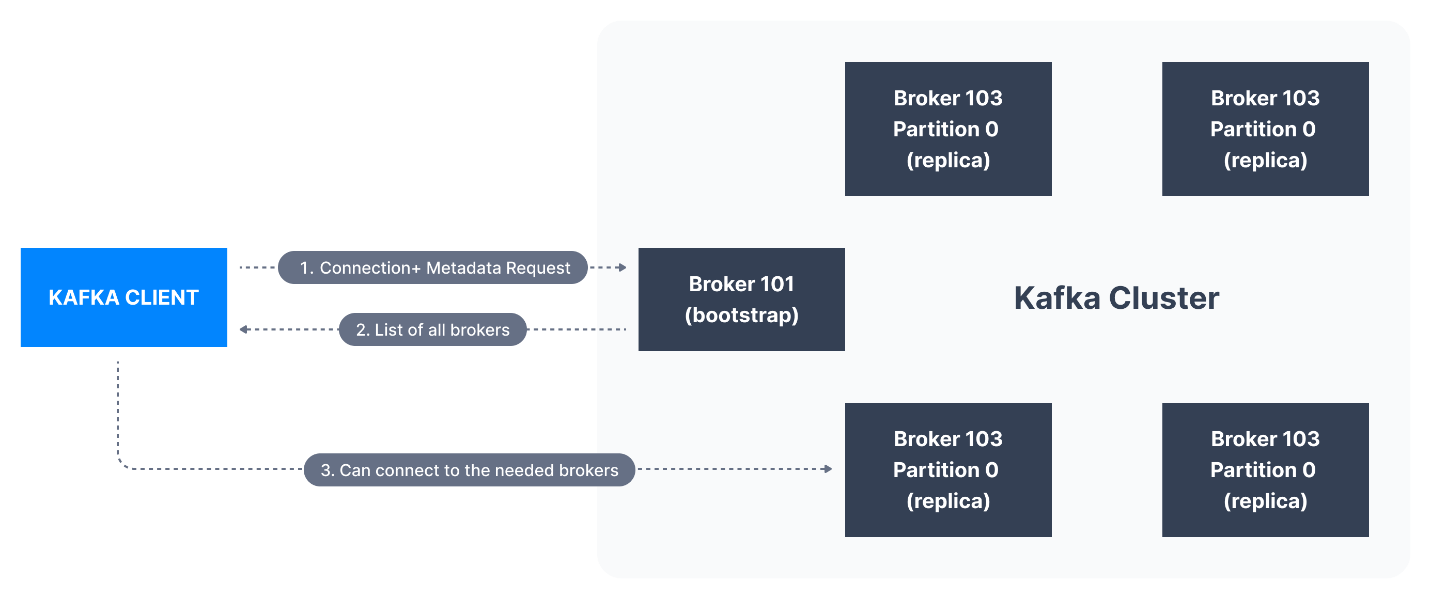
**PARTITION BROKER MAPPING**

Kafka does a good job of distributing **partitions** evenly among the available **brokers**. there may be fewer (or more) partitions of a topic than the number of brokers in the cluster. *Topic-B*, in our case, has two partitions only. In this case, *Broker 103* does not contain any partition of *Topic-B*.



A client that wants to send or receive messages from the Kafka cluster **may connect to any broker in the cluster.** Every broker in the cluster has metadata about all the other brokers and will help the client connect to them as well, and **therefore any broker in the cluster is also called a bootstrap server.**

The bootstrap server will return metadata to the client that consists of a list of all the brokers in the cluster. Then, when required, the client will know which exact broker to connect to to send or receive data, and accurately find which brokers contain the relevant topic-partition.



In practice, it is common for the Kafka client to reference at least two bootstrap servers in its connection URL, in the case one of them not being available, the other one should still respond to the connection request. That means that Kafka clients (and developers / DevOps) do not need to be aware of every single hostname of every single broker in a Kafka cluster, but only to be aware and reference two or three in the connection string for clients.

Horizontal scaling in Kafka:

The data and your partitions is going to be distributed across all brokers,

and this is what makes Kafka scale,

and what's actually called horizontal scaling,

because the more partitions and the more brokers we add,

the more the data is going to be spread out

across our entire cluster.

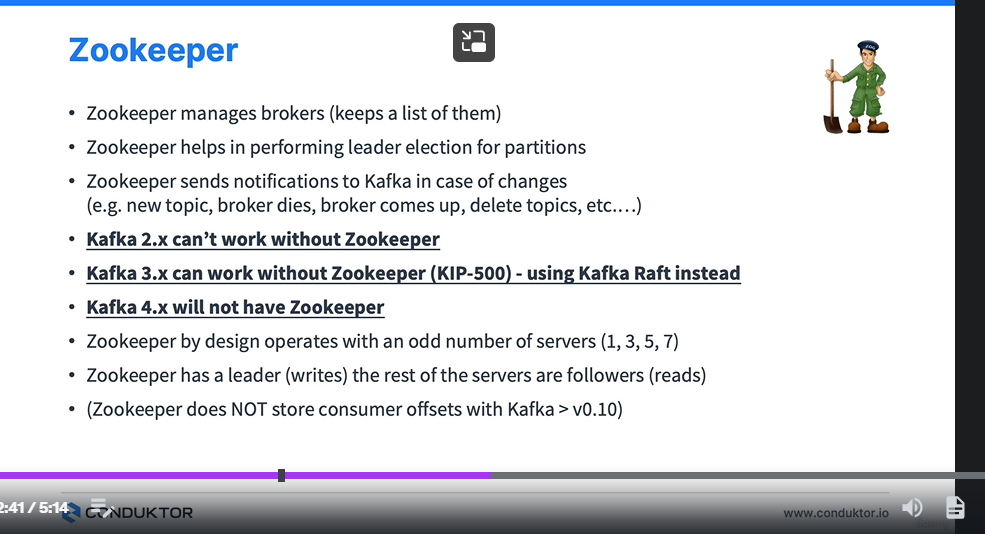
if you choose a replication factor of N, N being a number,

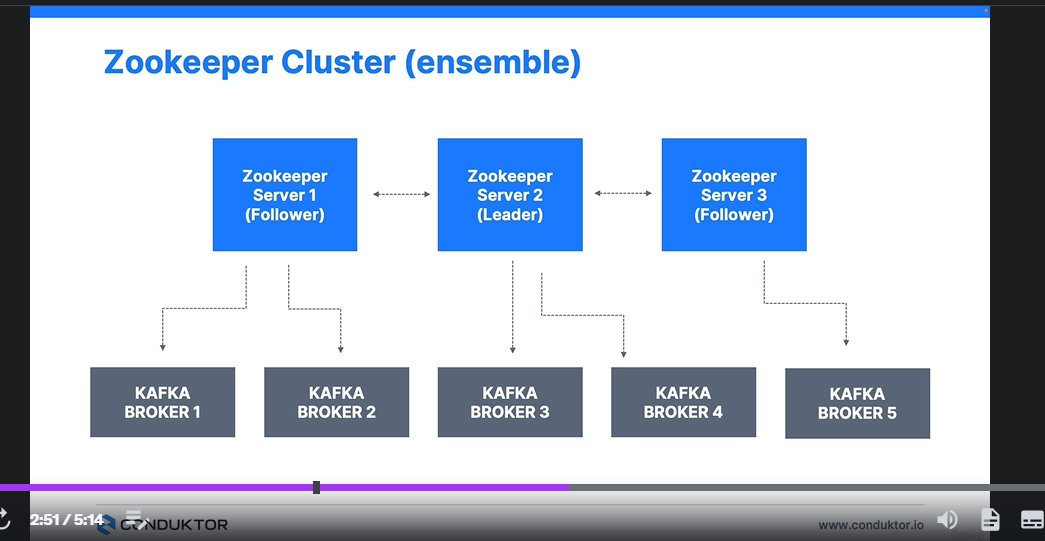
then you can permanently lose up to N-1

broker and still have a copy

of your data somewhere in your cluster.

And Kafka 2.X cannot work without Zookeeper.





So Kafka clients over time,

they have been migrated to leverage the brokers

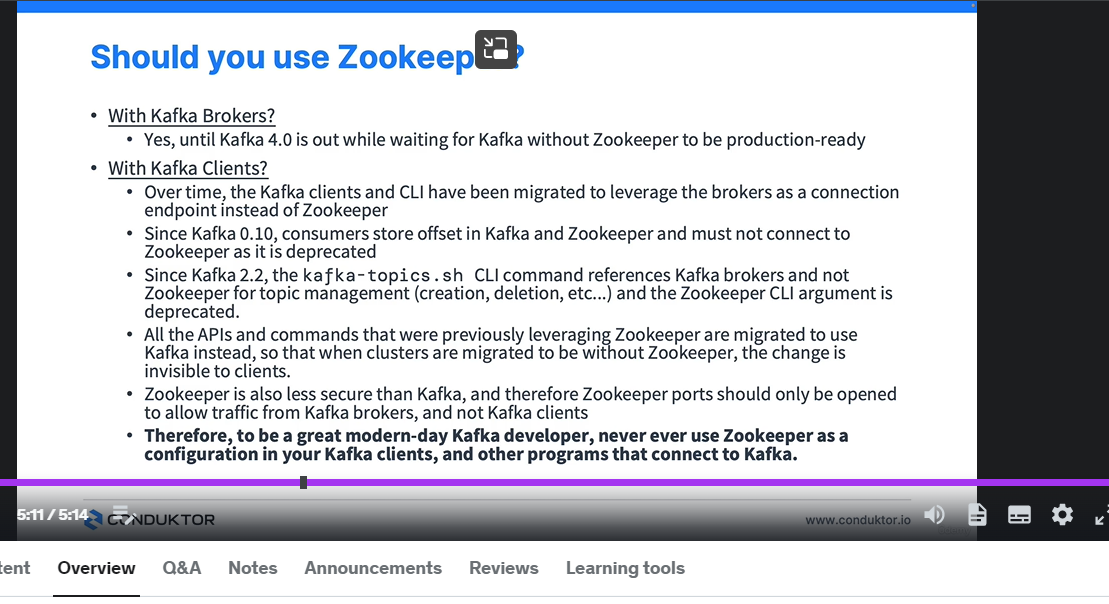
as the only connection endpoint instead of Zookeeper.

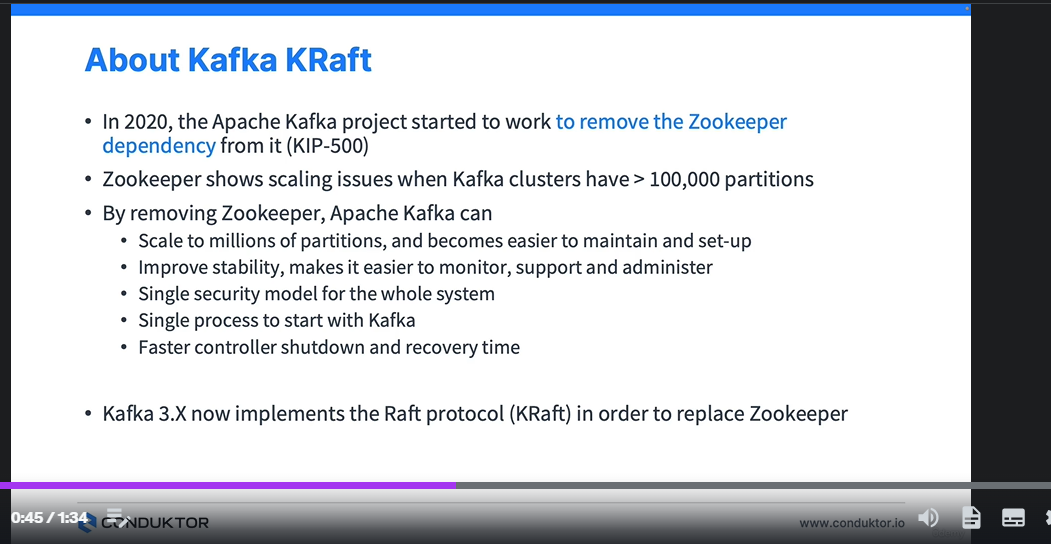
But before you used to connect your producer to Zookeeper

you used to connect your consumer to Zookeeper.

You used to connect your administration client

to Zookeeper and so on.





So with Zookeeper, we have this architecture,

with three Zookeepers for example,

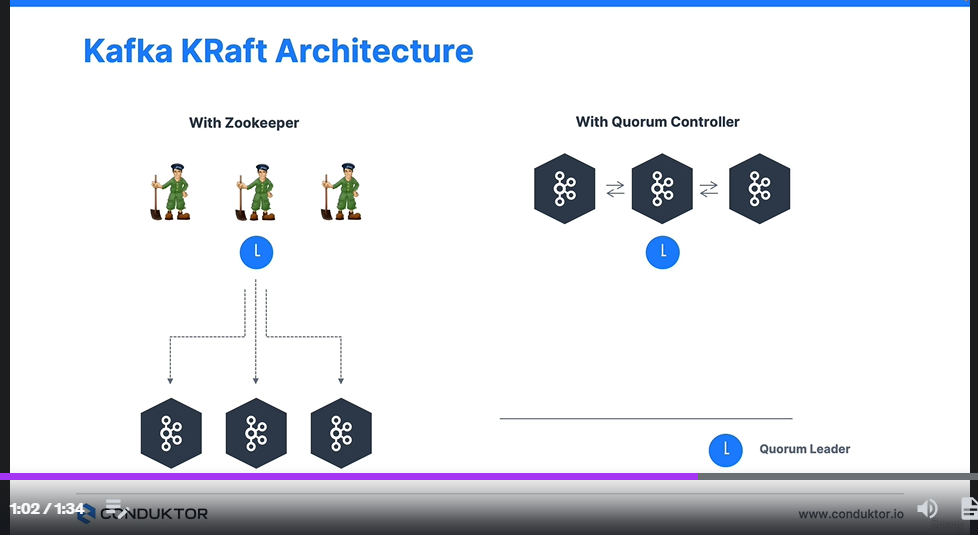
managing three Kafka brokers.

And with the KRaft mode,

we're only going to have three Kafka brokers.

One of them being designated as the **Quorum leader**

to replace the Zookeeper function.



Graphical user interface, application

Description automatically generated with medium confidence

**Kafka Theory Summary**

So we've seen that a Kaf Cluster

can be made of multiple brokers,

for example, I have nine brokers in my cluster.

And we've seen within the cluster,

we've seen the concept of topics, partitions, replication,

we've seen the partition leader and in-sync replicas

and we've seen the Kafka internal offset topic.

Now for producers, we've seen that they take data

from the source system and whatever you want

and then they send data into a batching Kafka.

We've seen the concept that round robin,

that means that the data is distributed

across all partitions in a topic.

Key based ordering, that means that when we specify a key,

then the same key is going to end up in the same partition.

And then acks strategy,

so we had an introduction to acks equal zero, one and all,

to discuss how much acknowledgement we want

when we write to a Kafka Cluster.

Okay, next we've seen the consumers,

so we've seen how consumers operate in the consumer group,

how they store offsets into a consumer offsets topic,

we've seen different conception modes such as at least one,

at most once or exactly once.

And then finally we've seen that Kaf Cluster

is managed right now by Zookeeper,

where there is leader follower concept in Zookeeper,

as well as broker management, metadata management.

And we've also seen how the fact

that the community is transitioning from using zookeeper

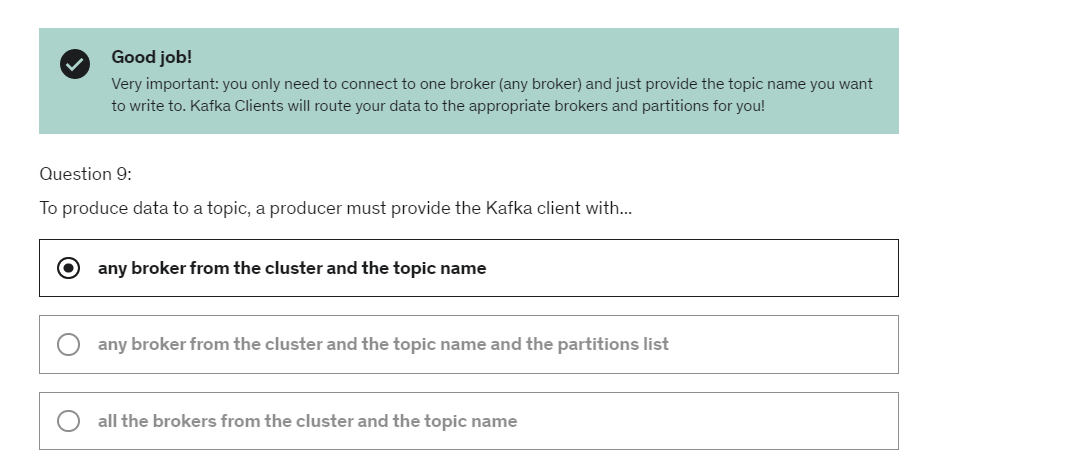
to using just a Kaf Cluster in craft mode, okay.

Graphical user interface, text, application, email

Description automatically generated

Timeline

Description automatically generated



Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application

Description automatically generated