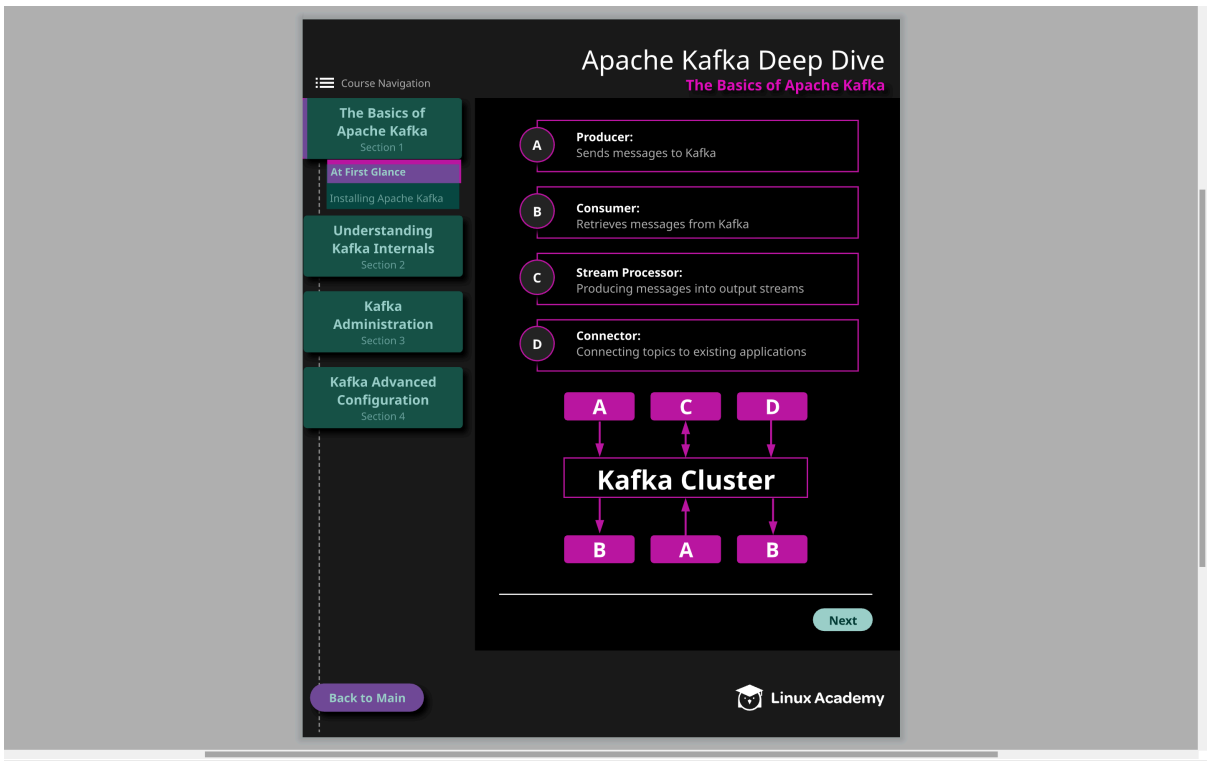
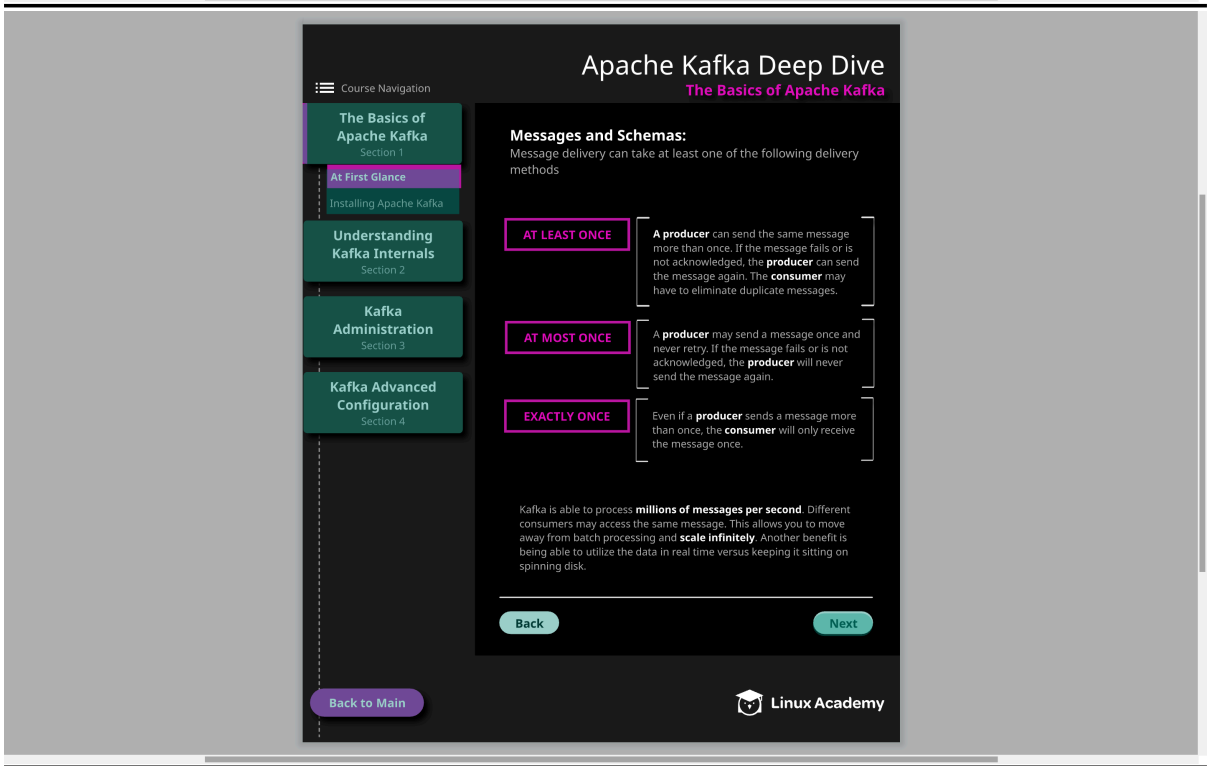
Kafka Linux Academy Basics



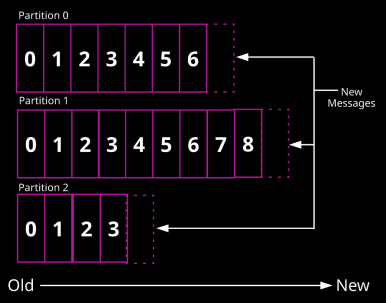


Apache Kafka Deep Dive

The Basics of Apache Kafka

Topics and Partitions:

Messages in Kafka are called **Topics**. Topics are divided into partitions, with each message receiving an incremental ID called the ***offset***.



When a new message is written:

- By default, it is kept for **one week**
- It **cannot** be altered or changed in any way
- The ID will increment **infinitely** (never start over at zero)
- It is randomly assigned to a partition, unless a **key** is provided

Course Navigation

The Basics of Apache Kafka
Section 1
At First Glance
Installing Apache Kafka

Understanding Kafka Internals
Section 2

Kafka Administration
Section 3

Kafka Advanced Configuration
Section 4

Apache Kafka Deep Dive

The Basics of Apache Kafka

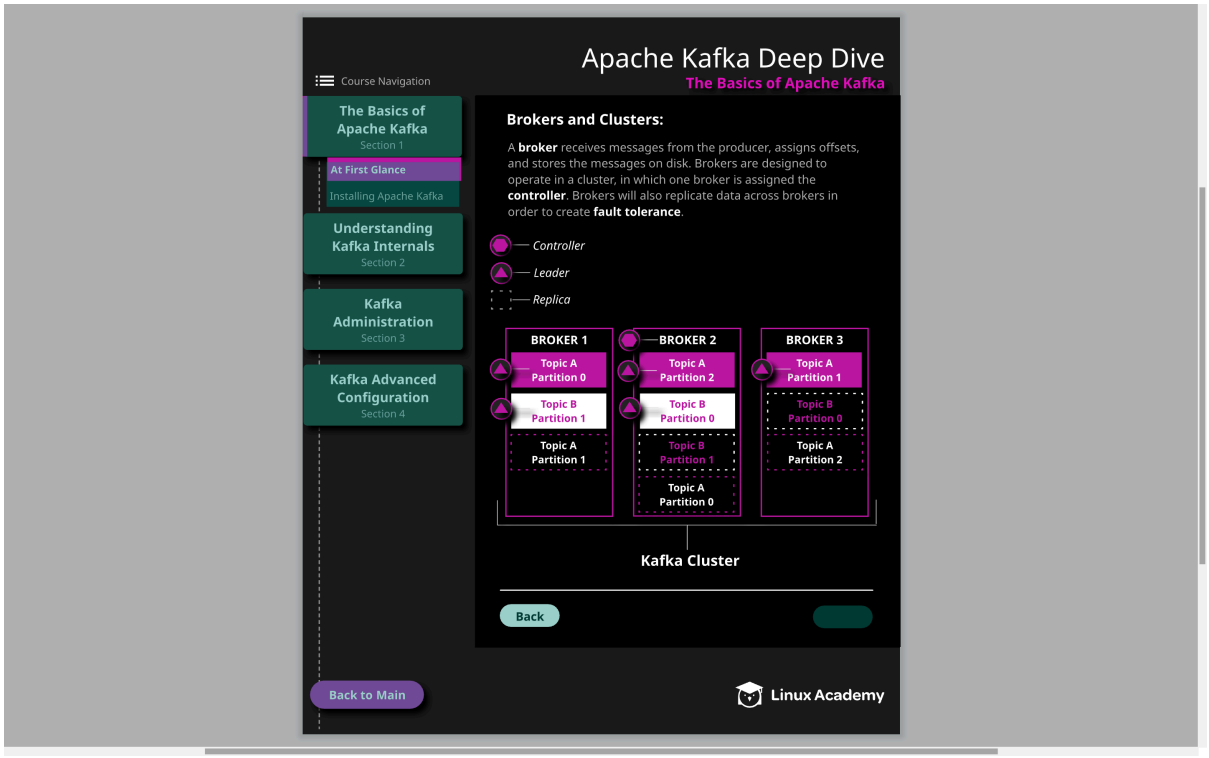
Producers and Consumers:

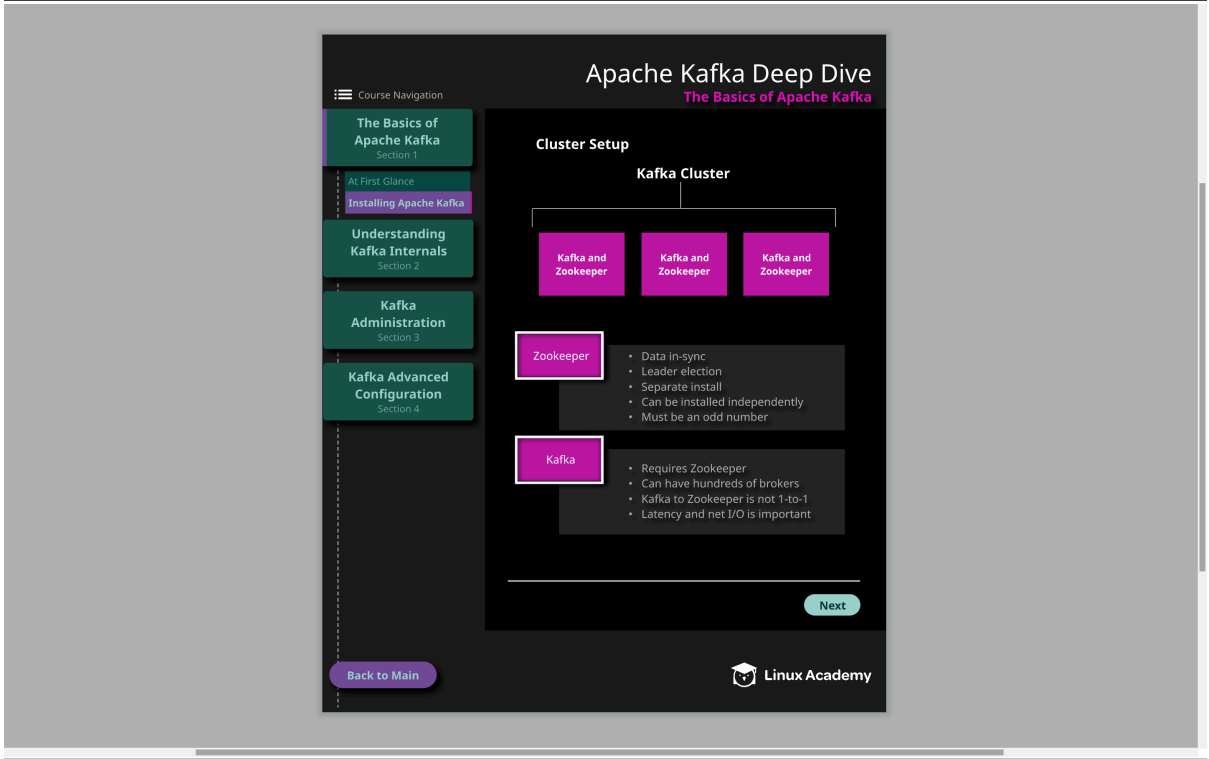
A **producer** is a client that writes data to the Kafka cluster in order to eventually get consumed. A consumer is the application that consumes or reads the messages. The consumer subscribes to one or more topics and reads the messages in the order in which they were produced.

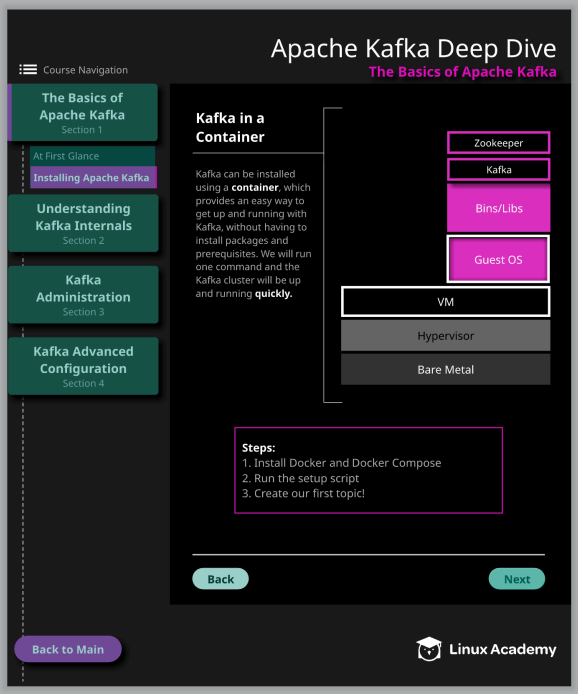
Acks:
Producers can choose whether to receive a confirmation of delivery by setting "acks" (acknowledgements)

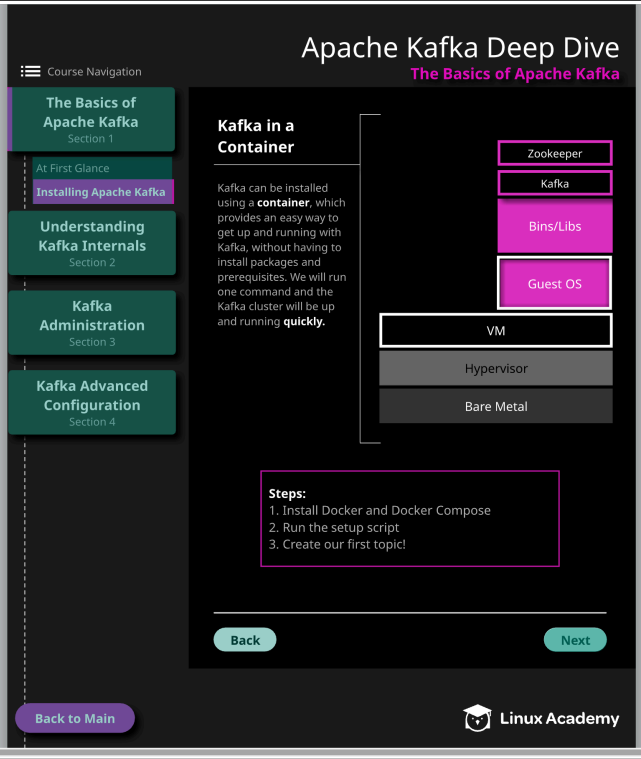
Key:
Producers specify a key, indicating that a message will go to the same partition every time

Consumer Group:
To ensure multiple consumers aren't reading the same message, consumer groups map reads to consumers.


Apache Kafka Deep Dive
The Basics of Apache Kafka
Course Navigation
The Basics of Apache Kafka
Section 1
At First Glance
Installing Apache Kafka
Understanding Kafka Internals
Section 2
Kafka Administration
Section 3
Kafka Advanced Configuration
Section 4
Brokers and Clusters:
A broker receives messages from the producer, assigns offsets, and stores the messages on disk. Brokers are designed to operate in a cluster, in which one broker is assigned the controller. Brokers will also replicate data across brokers in order to create fault tolerance.
Controller
Leader
Replica
BROKER 1
Topic A Partition 0
Topic B Partition 1
Topic A Partition 1
BROKER 2
Topic A Partition 2
Topic B Partition 0
Topic B Partition 1
Topic A Partition 0
BROKER 3
Topic A Partition 1
Topic B Partition 0
Topic A Partition 2
Kafka Cluster
Back
Back to Main
Linux Academy


Apache Kafka Deep Dive
The Basics of Apache Kafka
Course Navigation
The Basics of Apache Kafka
Section 1
At First Glance
Installing Apache Kafka
Understanding Kafka Internals
Section 2
Kafka Administration
Section 3
Kafka Advanced Configuration
Section 4
Cluster Setup
Kafka Cluster
Kafka and Zookeeper
Kafka and Zookeeper
Kafka and Zookeeper
Zookeeper
Data in-sync
Leader election
Separate install
Can be installed independently
Must be an odd number
Kafka
Requires Zookeeper
Can have hundreds of brokers
Kafka to Zookeeper is not 1-to-1
Latency and net I/O is important
Next
Back to Main
Linux Academy


Apache Kafka Deep Dive
The Basics of Apache Kafka
Course Navigation
The Basics of Apache Kafka
Section 1
At First Glance
Installing Apache Kafka
Understanding Kafka Internals
Section 2
Kafka Administration
Section 3
Kafka Advanced Configuration
Section 4
Kafka in a Container
Kafka can be installed using a **container**, which provides an easy way to get up and running with Kafka, without having to install packages and prerequisites. We will run one command and the Kafka cluster will be up and running **quickly.**
Zookeeper
Kafka
Bins/Libs
Guest OS
VM
Hypervisor
Bare Metal
Steps:
1. Install Docker and Docker Compose
2. Run the setup script
3. Create our first topic!
Back
Next
Back to Main
Linux Academy


Apache Kafka Deep Dive
The Basics of Apache Kafka
Course Navigation
The Basics of Apache Kafka
Section 1
At First Glance
Installing Apache Kafka
Understanding Kafka Internals
Section 2
Kafka Administration
Section 3
Kafka Advanced Configuration
Section 4
Kafka in a Container
Kafka can be installed using a **container**, which provides an easy way to get up and running with Kafka, without having to install packages and prerequisites. We will run one command and the Kafka cluster will be up and running **quickly.**
Zookeeper
Kafka
Bins/Libs
Guest OS
VM
Hypervisor
Bare Metal
Steps:
1. Install Docker and Docker Compose
2. Run the setup script
3. Create our first topic!
Back
Next
Back to Main
Linux Academy

Course Navigation
The Basics of Apache Kafka
Section 1
At First Glance
Installing Apache Kafka
Understanding Kafka Internals
Section 2
Kafka Administration
Section 3
Kafka Advanced Configuration
Section 4

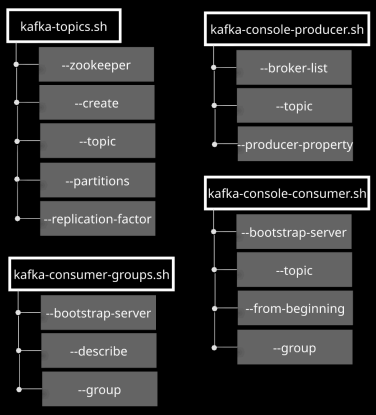
Installing Kafka using Binaries

Kafka and Zookeeper are independent of each other. They can operate on different VMs, but for the purposes of this course, we'll be installing them together using their **binaries**, installing them on three different cloud servers. This will allow for high-availability and fault-tolerance.

Steps:
1. Download the Binaries
2. Configure Zookeper and Start Service
3. Configure Kafka and Start Service
4. Create our first topic!

Apache Kafka Deep Dive
The Basics of Apache Kafka
Course Navigation
The Basics of Apache Kafka
Section 1
At First Glance
Installing Apache Kafka
Understanding Kafka Internals
Section 2
Kafka Administration
Section 3
Kafka Advanced Configuration
Section 4

Breaking Down the Commands



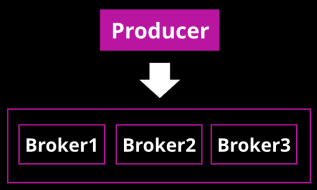
Back
Next
Back to Main
Linux Academy

Course Navigation
The Basics of Apache Kafka
Section 1
At First Glance
Installing Apache Kafka
Understanding Kafka Internals
Section 2
Kafka Administration
Section 3
Kafka Advanced Configuration
Section 4

Kafka Producer

```
./bin/kafka-console-consumer.sh \
--broker-list kafka1:9092 --topic test
```

A **Producer** sends messages to the Kafka cluster. Producers are a required component to getting messages into Kafka. A Producer is also a way to send messages inside of Kafka itself. For example, when sending a message from one topic to another inside the same cluster.

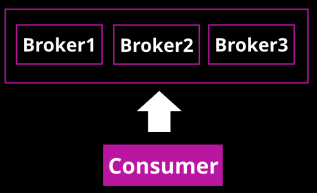


Apache Kafka Deep Dive
The Basics of Apache Kafka
Course Navigation
The Basics of Apache Kafka
Section 1
At First Glance
Installing Apache Kafka
Understanding Kafka Internals
Section 2
Kafka Administration
Section 3
Kafka Advanced Configuration
Section 4

Kafka Consumer

```
./bin/kafka-console-consumer.sh \
--bootstrap-server kafka1:9092 --topic test
```

A **Consumer** retrieves messages from Kafka. It is the only way to get messages out of Kafka. Consuming applications will subscribe to a topic and continually poll for new data. The messages will be consumed in order and the consumer will never re-read the same message. Instead, it will keep track of its place and subscribe to only new messages.



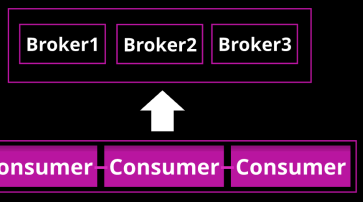
Back
Next
Back to Main
Linux Academy

Course Navigation
The Basics of Apache Kafka
Section 1
At First Glance
Installing Apache Kafka
Understanding Kafka Internals
Section 2
Kafka Administration
Section 3
Kafka Advanced Configuration
Section 4

Kafka Consumer Group

```
./bin/kafka-console-consumer.sh \ --bootstrap-server
kafka1:9092 --topic test --group application1
```

A **Consumer Group** allows for multiple consumers to process the data from one topic, all without reading the same message. Kafka was designed to have multiple consumers, as it is a faster and more efficient way to process data. Consumer groups act independently and there is no limit to how many consumers you can have in a group.

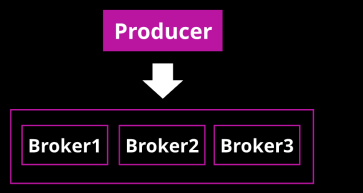


Apache Kafka Deep Dive
The Basics of Apache Kafka
Course Navigation
The Basics of Apache Kafka
Section 1
At First Glance
Installing Apache Kafka
Understanding Kafka Internals
Section 2
Kafka Administration
Section 3
Kafka Advanced Configuration
Section 4

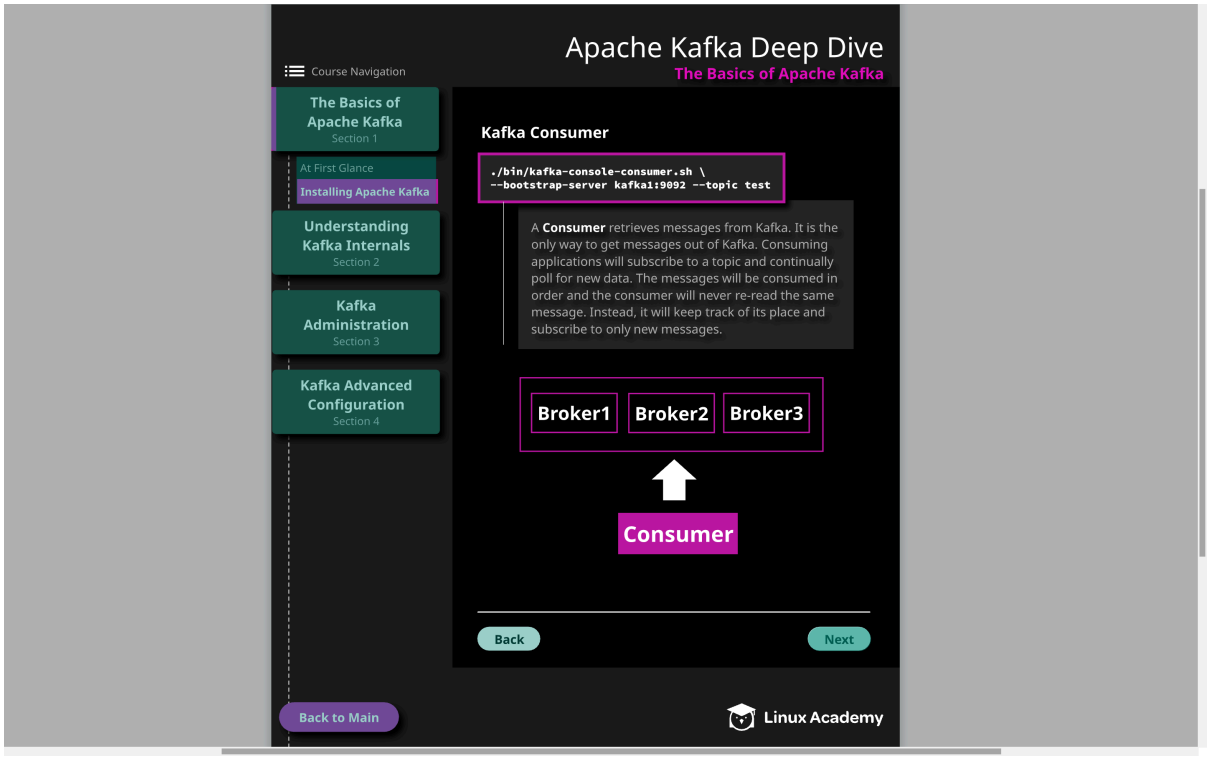
Kafka Producer

```
./bin/kafka-console-consumer.sh \
--broker-list kafka1:9092 --topic test
```

A **Producer** sends messages to the Kafka cluster. Producers are a required component to getting messages into Kafka. A Producer is also a way to send messages inside of Kafka itself. For example, when sending a message from one topic to another inside the same cluster.

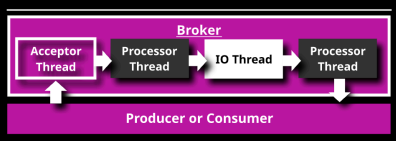


Back
Next
Back to Main
Linux Academy


Course Navigation
Apache Kafka Deep Dive
The Basics of Apache Kafka
The Basics of Apache Kafka
Section 1
At First Glance
Installing Apache Kafka
Understanding Kafka Internals
Section 2
Kafka Administration
Section 3
Kafka Advanced Configuration
Section 4
Kafka Consumer
./bin/kafka-console-consumer.sh \
--bootstrap-server kafka1:9092 --topic test
A Consumer retrieves messages from Kafka. It is the only way to get messages out of Kafka. Consuming applications will subscribe to a topic and continually poll for new data. The messages will be consumed in order and the consumer will never re-read the same message. Instead, it will keep track of its place and subscribe to only new messages.
Broker1
Broker2
Broker3
Consumer
Back
Next
Back to Main
Linux Academy

Course Navigation
The Basics of Apache Kafka
Section 1
Understanding Kafka Internals
Section 2
Taking a Closer Look
Data Delivery
Producers & Consumers
Kafka Administration
Section 3
Kafka Advanced Configuration
Section 4

Handling Requests



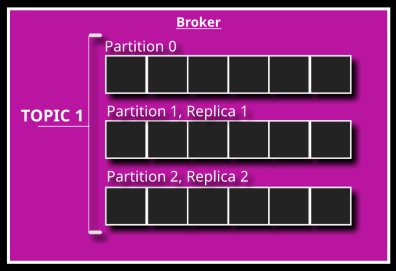
Acceptor Thread	Creates the connection from client to Broker.
Processor Thread	Takes requests from clients and places them into the request queue.
IO Thread	Processes the requests in the requests queue.
Request Queue	Requests waiting to be processed.
Response Queue	Requests waiting to be sent back to the clients.

Back
Next

Apache Kafka Deep Dive
Understanding Kafka Internals

Course Navigation
The Basics of Apache Kafka
Section 1
Understanding Kafka Internals
Section 2
Taking a Closer Look
Data Delivery
Producers & Consumers
Kafka Administration
Section 3
Kafka Advanced Configuration
Section 4

Partitions



- Partitions within a topic are split between brokers.
- The partitions themselves will never be split up.
- The partitions will continue to get larger as the number of messages grow.
- Messages will never be removed from the partition, only appended.
- Messages are stored in the directory specified in the `server.properties` file (`log.dirs`).

Back

The Basics of Apache Kafka
Section 1

Understanding Kafka Internals
Section 2

Taking a Closer Look
Data Delivery
Producers & Consumers

Kafka Administration
Section 3

Kafka Advanced Configuration
Section 4

Developing Applications for Kafka:

We are going to be using **Maven** to manage our Java project packages. We will edit the `pom.xml` file, and create a **producer** class.

```
kafka-app
├── alert.avsc
├── pom.xml
├── src
│   ├── main
│   │   └── java
│   │       └── com
│   │           └── github
│   │               └── chadmcrowell
│   │                   └── Producer.java
│   └── test
│       └── java
│           └── com
│               └── github
│                   └── chadmcrowell
│                       └── AppTest.java
└── target
    ├── classes
    │   └── com
    │       └── github
    │           └── chadmcrowell
    │               ├── Consumer.class
    │               └── Producer.class
    ├── maven-status
    │   └── maven-compiler-plugin
    │       ├── compile
    │       │   └── default-compile
    │       │       ├── createdFiles.lst
    │       │       └── inputFiles.lst
    │       └── testCompile
    │           └── default-testCompile
    │               ├── createdFiles.lst
    │               └── inputFiles.lst
    └── test-classes
```


The Basics of Apache Kafka
Section 1

Understanding Kafka Internals
Section 2

Kafka Administration
Section 3

Topic Administration
Storage Administration
Stream Processing
Data Replication
Monitoring

Kafka Advanced Configuration
Section 4

Topic Tools

```
bin/kafka-topics.sh --zookeeper zookeeper1:2181/kafka
--create
--topic topic-1
--replication-factor 1
--partitions 3
--if-not-exists

bin/kafka-topics.sh --zookeeper zookeeper1:2181/kafka
--alter
--topic topic-1
--partitions 7

bin/kafka-topics.sh --zookeeper zookeeper1:2181/kafka
--delete
--topic topic-1

bin/kafka-topics.sh --zookeeper zookeeper1:2181/kafka
--list

bin/kafka-topics.sh --zookeeper zookeeper1:2181/kafka
--describe
--topics-with-overrides
--under-replicated-partitions
--unavailable-partitions

bin/kafka-configs.sh --zookeeper zookeeper1:2181/kafka
--describe
--entity-type topics
--entity-name topic-1

bin/kafka-configs.sh --zookeeper zookeeper1:2181/kafka
--alter
--entity-type topics
--entity-name connect-test
--add-config retention.ms=3600000

bin/kafka-configs.sh --zookeeper zookeeper1:2181/kafka
--entity-type brokers
--entity-name 0
--alter
--delete-config log.cleaner.threads
```

Course Navigation
The Basics of Apache Kafka
Section 1
Understanding Kafka Internals
Section 2
Kafka Administration
Section 3
Topic Administration
Storage Administration
Stream Processing
Data Replication
Monitoring
Kafka Advanced Configuration
Section 4

Topic Configurations

Consider a situation where you approach established Kafka clusters with millions of committed messages. You could modify the groups where topics are associated. You can do this by using the **`kafka-consumer-groups.sh`** to reset the offsets or delete a group entirely.

```
bin/kafka-consumer-groups.sh --bootstrap-server kafka1:9092
--list
--describe
--group application1
--partitions 3
--members
--verbose
--state
--delete
--reset-offsets
--topic topic-1
--to-latest
```

Back Next

Apache Kafka Deep Dive
Kafka Administration

Course Navigation
The Basics of Apache Kafka
Section 1
Understanding Kafka Internals
Section 2
Kafka Administration
Section 3
Topic Administration
Storage Administration
Stream Processing
Data Replication
Monitoring
Kafka Advanced Configuration
Section 4

Message Behavior

As an administrator, there may be times where you have to manually adjust the partitions and messages. As your Kafka cluster gets larger, the maintenance of that cluster causes messages to consume the broker over time. In this case, you must take actions to restore the health of the cluster.

A **Offset:** See if consumer offsets are committed.

B **Line Reading:** Provide your own class for reading lines.

C **Move Partitions:** Move partitions if the broker fails in the middle of a reassignment.

Back

Back to Main

Course Navigation
The Basics of Apache Kafka
Section 1
Understanding Kafka Internals
Section 2
Kafka Administration
Section 3
Topic Administration
Storage Administration
Stream Processing
Data Replication
Monitoring
Kafka Advanced Configuration
Section 4

File Formats and Indexes

Log Segments | /data/kafka

Within our message logs, we'll see the individual segment files accrue as messages are produced. To maintain this over time, you can set file compaction. You can also delete indexes, and they will populate automatically.

Index:
`/data/kafka/test-0/00000000000000.index`
LogSegment:
`/data/kafka/test-0/00000000000000.log`
Compaction:
`/data/kafka/test-0/006.timeindex.deleted`

Next

Apache Kafka Deep Dive
Kafka Administration

Course Navigation
The Basics of Apache Kafka
Section 1
Understanding Kafka Internals
Section 2
Kafka Administration
Section 3
Topic Administration
Storage Administration
Stream Processing
Data Replication
Monitoring
Kafka Advanced Configuration
Section 4

File Management

Segment 0	Segment 1	Segment 2	Segment 3	Segment 3
Offset 0-95	Offset 96-194	Offset 195-293	Offset 294-392	ACTIVE

Time Retention	Size Retention
log.retention.hours	log.retention.bytes
default is 168	default is 1 GB

The segment we are currently writing to is called the **active segment**. By default, the segment is deleted when it reaches a size of **1 GB** or has been inactive for **1 week**, whichever comes first.

The Broker has an open file handle to each segment in every partition, even inactive segments. Make sure your operating system can handle so many open files at once.

Back | Next

Apache Kafka Deep Dive

Kafka Administration

Course Navigation

The Basics of Apache Kafka
Section 1

Understanding Kafka Internals
Section 2

Kafka Administration
Section 3

Topic Administration
Storage Administration
Stream Processing
Data Replication
Monitoring

Kafka Advanced Configuration
Section 4

Storage Structures

1. **Batch Processing and Operational Workloads**
   - **Lambda Architecture:** View real-time and historical data. This adds difficulty due to the many simultaneous interfaces.
2. **Version 1 and Version 2 in Parallel**
   - **Kappa Architecture:** Switch your view as a cut-over. This helps with migration or transitions to continuous jobs.
3. **Multi-Cluster**
   - **Multiple Consumption:** Replicate clusters for scale, ingesting data from different Kafka clusters. The potential upside is bandwidth savings.

Back

Apache Kafka Deep Dive

Kafka Administration

Course Navigation

The Basics of Apache Kafka
Section 1

Understanding Kafka Internals
Section 2

Kafka Administration
Section 3

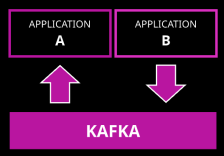
Topic Administration
Storage Administration
Stream Processing
Data Replication
Monitoring

Kafka Advanced Configuration
Section 4

How Streams Work

A stream is a sequence of events. Because Kafka does not rely on an external process framework, any type of stream can exist.

- Credit card transactions
- Stock trades
- Package deliveries
- Network events
- Sensor events in manufacturing equipment
- Emails sent
- Moves in a game



Next

Back to Main

Linux Academy

The Basics of Apache Kafka
Section 1

Understanding Kafka Internals
Section 2

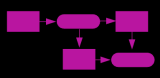
Kafka Administration
Section 3

Topic Administration
Storage Administration
Stream Processing
Data Replication
Monitoring

Kafka Advanced Configuration
Section 4

Design Patterns

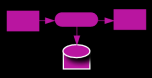
1. Single Event Processing



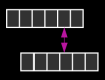
2. Local State Processing

3. Multiphase Processing

4. External Processing



5. Windowed Join



6. Out of Sequence Events

Reprocessing - Migrating to a new version of a streaming application.

Back | Next

Course Navigation

Apache Kafka Deep Dive
Kafka Administration

The Basics of Apache Kafka
Section 1

Understanding Kafka Internals
Section 2

Kafka Administration
Section 3

Topic Administration
Storage Administration
Stream Processing
Data Replication
Monitoring

Kafka Advanced Configuration
Section 4

Frameworks

Deciding which framework depends on the type of application you plan to write. Different applications require different stream-processing solutions.

Ingest:
Gets data from one system to the other.

Low Millisecond:
Works for applications requiring quick response.

Real-Time Data Analytics:
Supports performing complex aggregations of data to gain insight.

Asynchronous Microservices:
Performs a single action to serve a single need. Requires local state caching events.

NOTE:
When choosing a framework, ensure it fits well with your existing infrastructure. Additional considerations include scaling, repairing, efficiency with APIs, and longevity. Taking careful consideration to establish a great foundation will save you many days on unnecessary troubleshooting.

Back

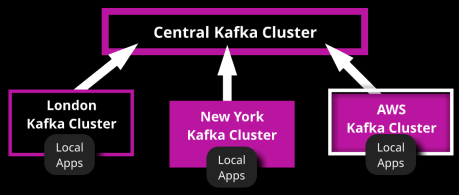
Back to Main

Linux Academy

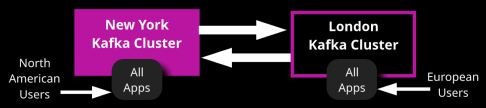
Course Navigation
The Basics of Apache Kafka
Section 1
Understanding Kafka Internals
Section 2
Kafka Administration
Section 3
Topic Administration
Storage Administration
Stream Processing
Data Replication
Monitoring
Kafka Advanced Configuration
Section 4

Multi-Cluster Architectures:

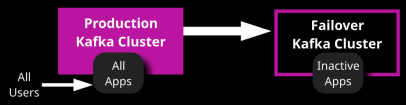
Hub-And-Spoke



Active-Active



Active-Standby



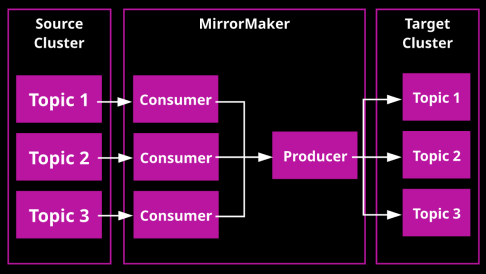
Next

Apache Kafka Deep Dive
Kafka Administration

Course Navigation
The Basics of Apache Kafka
Section 1
Understanding Kafka Internals
Section 2
Kafka Administration
Section 3
Topic Administration
Storage Administration
Stream Processing
Data Replication
Monitoring
Kafka Advanced Configuration
Section 4

MirrorMaker

You use MirrorMaker for replicating data between two data centers. It's a collection of consumers in a consumer group. The group reads data from the set of topics you specify. Then MirrorMaker creates the thread and sends it to the target cluster. It creates one thread per consumer.



Configuration:
MirrorMaker has a number of configuration options. You typically install it as a service and run it at the destination data center. The biggest task is monitoring lag to ensure the destination cluster is not falling behind the source.

Back

Course Navigation
The Basics of Apache Kafka
Section 1
Understanding Kafka Internals
Section 2
Kafka Administration
Section 3
Topic Administration
Storage Administration
Stream Processing
Data Replication
Monitoring
Kafka Advanced Configuration
Section 4

Cluster and Broker Monitoring

Metrics can be accessed from:

- Java Management Extensions (JMX)
- Yammer Metrics

```
get /brokers/ids/3
```

***Use an Intelligent Platform Management Interface (IPMI) to monitor hardware health.**

Monitor the Overall Health

Problems:

- Under-replicated partitions
- Hardware failures

Solutions:

- Check for under-replicated partitions.

```
bin/kafka-preferred-replica-election.sh --bootstrap-server kafka1:9092 \
--path-to-json-file topicPartitionList.json
```

- Run a preferred replica election.

```
bin/kafka-topics.sh --zookeeper zookeeper1:2181/kafka \
--describe --under-replicated-partitions
```

- Move partition(s) to a different broker.

```
bin/kafka-reassign-partitions.sh --zookeeper zookeeper1:2181/kafka \
--execute --reassignment-json-file replicacount.json
```

Apache Kafka Deep Dive
Kafka Administration

Course Navigation
The Basics of Apache Kafka
Section 1
Understanding Kafka Internals
Section 2
Kafka Administration
Section 3
Topic Administration
Storage Administration
Stream Processing
Data Replication
Monitoring
Kafka Advanced Configuration
Section 4

The Broker Metrics to Follow:

- ACTIVE CONTROLLER COUNT — Is the broker the controller?
- REQUEST HANDLER IDLE RATIO — How much load is the broker under?
- ALL TOPICS BYTES IN — Do I need to scale up the number of brokers?
- ALL TOPICS BYTES OUT — How high is consumer traffic?
- ALL TOPICS MESSAGES IN — How many messages per second?
- PARTITION COUNT — How many partitions are assigned to a broker?
- LEADER COUNT — How many partitions is this broker a leader for?
- OFFLINE PARTITIONS — How many brokers have no leader?
- REQUEST METRICS — How many requests are going to the broker?

Apache Kafka Deep Dive

Kafka Administration

Course Navigation

- The Basics of Apache Kafka — Section 1
- Understanding Kafka Internals — Section 2
- Kafka Administration — Section 3
  - Topic Administration
  - Storage Administration
  - Stream Processing
  - Data Replication
  - Monitoring
- Kafka Advanced Configuration — Section 4

Java Monitoring

Garbage Collection Beans to Monitor:

```
java.lang:type=GarbageCollector,name=G1 Old Generation*
java.lang:type=GarbageCollector,name=G1 Young Generation*
```

* for Oracle Java 1.8 JRE with G1 GC

# of GC cycles:	`CollectionCount`
Time spent in GC cycle:	`CollectionTime`

JVM

The JVM can provide some OS information through the **`java.lang:type=OperatingSystem`** bean.

The two attributes we can collect are:

1. **`OpenFileDescriptorCount`**
2. **`MaxFileDescriptorCount`**

Apache Kafka Deep Dive

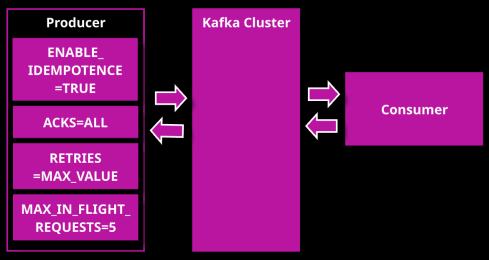
Kafka Advanced Configuration

Course Navigation

- The Basics of Apache Kafka — Section 1
- Understanding Kafka Internals — Section 2
- Kafka Administration — Section 3
- Kafka Advanced Configuration — Section 4
  - Advanced Producers
  - Advanced Consumers
  - Advanced Topics

Idempotent Producers

To eliminate the possibility of duplicate messages, you can set **`enable.idempotence`** to **`true`** and the consumer will delete duplicate messages .



Producer Configuration:
There are over 50 producer configurations. Kafka documentation indicates whether certain configurations are high importance or low. Idempotence is listed as low importance due to its effect on efficiency. However, you can use this to ensure your messages arrive in their entirety if safety is a concern.

The Basics of Apache Kafka
Section 1

Understanding Kafka Internals
Section 2

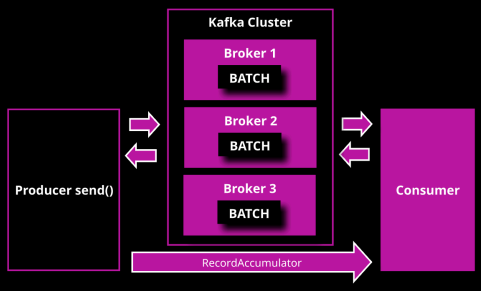
Kafka Administration
Section 3

Kafka Advanced Configuration
Section 4

Advanced Producers
Advanced Consumers
Advanced Topics

Batch Compression

Sending a batch as opposed to individual messages is very important in Kafka. Sending larger batches with compression will significantly improve throughput.



Batch Size and Timing:
When multiple records are sent to the same partition, the producer can batch them together. The **`batch.size`** configuration controls the amount of memory used for each batch. The **`linger.ms`** configuration will help increase the batch size to get the best compression and throughput.

Course Navigation

Apache Kafka Deep Dive
Kafka Advanced Configuration

The Basics of Apache Kafka
Section 1

Understanding Kafka Internals
Section 2

Kafka Administration
Section 3

Kafka Advanced Configuration
Section 4

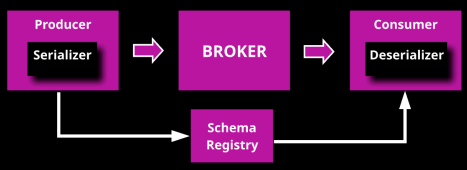
Advanced Producers
Advanced Consumers
Advanced Topics

Serializer

So far, we have been using a prebuilt serializer called **`StringSeralizer`**. But, what if we have a specific format in which we need to produce? Custom serialization can be used to translate your data in a format that can be stored, transmitted, and retrieved.

Avro Serializer:
`io.confluent.kafka.serializers.KafkaAvroSerializer`

String Serializer:
`org.apache.kafka.common.serialization.StringSerializer`



The Basics of Apache Kafka
Section 1

Understanding Kafka Internals
Section 2

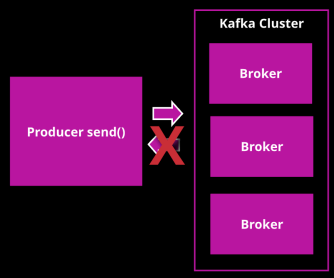
Kafka Administration
Section 3

Kafka Advanced Configuration
Section 4

Advanced Producers
Advanced Consumers
Advanced Topics

Producer Buffer Memory

If a producer is producing messages faster than the broker can receive those messages, the messages will be sent to a buffer memory on the producer.



max.block.ms:
If the producer is sending messages to the broker and the broker is not responding for 60 seconds, you will receive an exception error. This can occur if the broker's buffer is filled or the broker is down.

The Basics of Apache Kafka
Section 1

Understanding Kafka Internals
Section 2

Kafka Administration
Section 3

Kafka Advanced Configuration
Section 4

Advanced Producers
Advanced Consumers
Advanced Topics

Reading Duplicate Messages

`fetch.min.bytes` `fetch.max.bytes`	The minimum and maximum batch size of the request from the consumer. **The defaults are 1 byte and 50 MB.**
`max.poll.records`	The maximum number of messages the consumer receives when polling. **The default is 500.**
`max.partition.fetch.bytes`	The maximum amount of data the broker returns to the consumer. **The default is 1 MB.**

```
while(true){
    List<Records> batch = consumer.poll(Duration.ofMillis(100))
    doSomethingSynchronous(batch)
}
```

`enable.auto.commit=true`

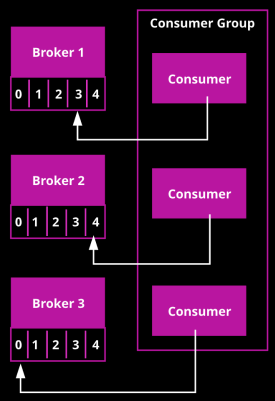
```
while(true){
    batch += consumer.poll(Duration.ofMillis(100))
    if isReady(batch) {
        doSomethingSyncronous(batch)
        consumer.commitSync();
    }
}
```

`enable.auto.commit=false`

The Basics of Apache Kafka
Section 1
Understanding Kafka Internals
Section 2
Kafka Administration
Section 3
Kafka Advanced Configuration
Section 4
Advanced Producers
Advanced Consumers
Advanced Topics

Tracking Offsets

When a consumer updates its current position in the partition, it's called a **commit**. A consumer produces a message to the **__consumer_offsets** topic. If the consumer crashes, it triggers a rebalance and the consumer may be assigned a new set of partitions.

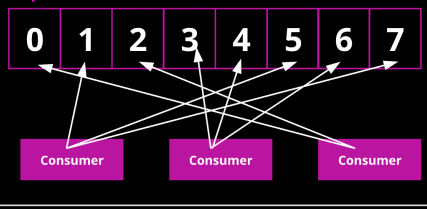


Course Navigation
Apache Kafka Deep Dive
Kafka Advanced Configuration

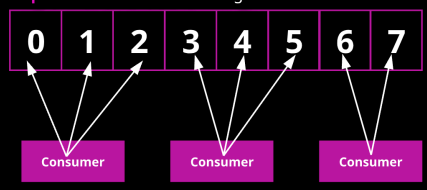
The Basics of Apache Kafka
Section 1
Understanding Kafka Internals
Section 2
Kafka Administration
Section 3
Kafka Advanced Configuration
Section 4
Advanced Producers
Advanced Consumers
Advanced Topics

Partition Rebalancing

Topic with 8 Partitions - Round Robin



Topic with 8 Partitions - Range



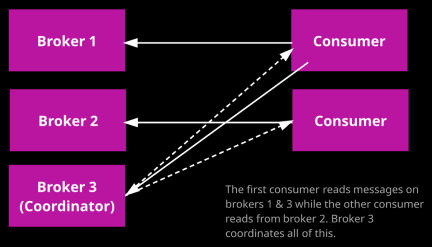
Moving partition ownership from one consumer to another is called a rebalance. This allows us to easily and safely add and remove consumers. Outside of adding or removing consumers, we don't want to rebalance.

Back
Next

Course Navigation
The Basics of Apache Kafka
Section 1
Understanding Kafka Internals
Section 2
Kafka Administration
Section 3
Kafka Advanced Configuration
Section 4
Advanced Producers
Advanced Consumers
Advanced Topics

Consumer Group Coordinator

One of the brokers in the Kafka cluster is established as a consumer group coordinator. That broker continuously reaches out to all consumers and checks if they have a heartbeat. This broker is also responsible for making the appropriate adjustments when a consumer fails or a new consumer joins the group.

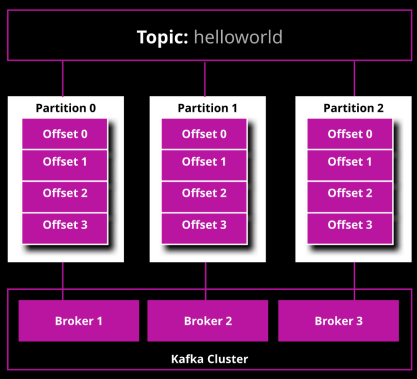


Back

Apache Kafka Deep Dive
Kafka Advanced Configuration

Course Navigation
The Basics of Apache Kafka
Section 1
Understanding Kafka Internals
Section 2
Kafka Administration
Section 3
Kafka Advanced Configuration
Section 4
Advanced Producers
Advanced Consumers
Advanced Topics

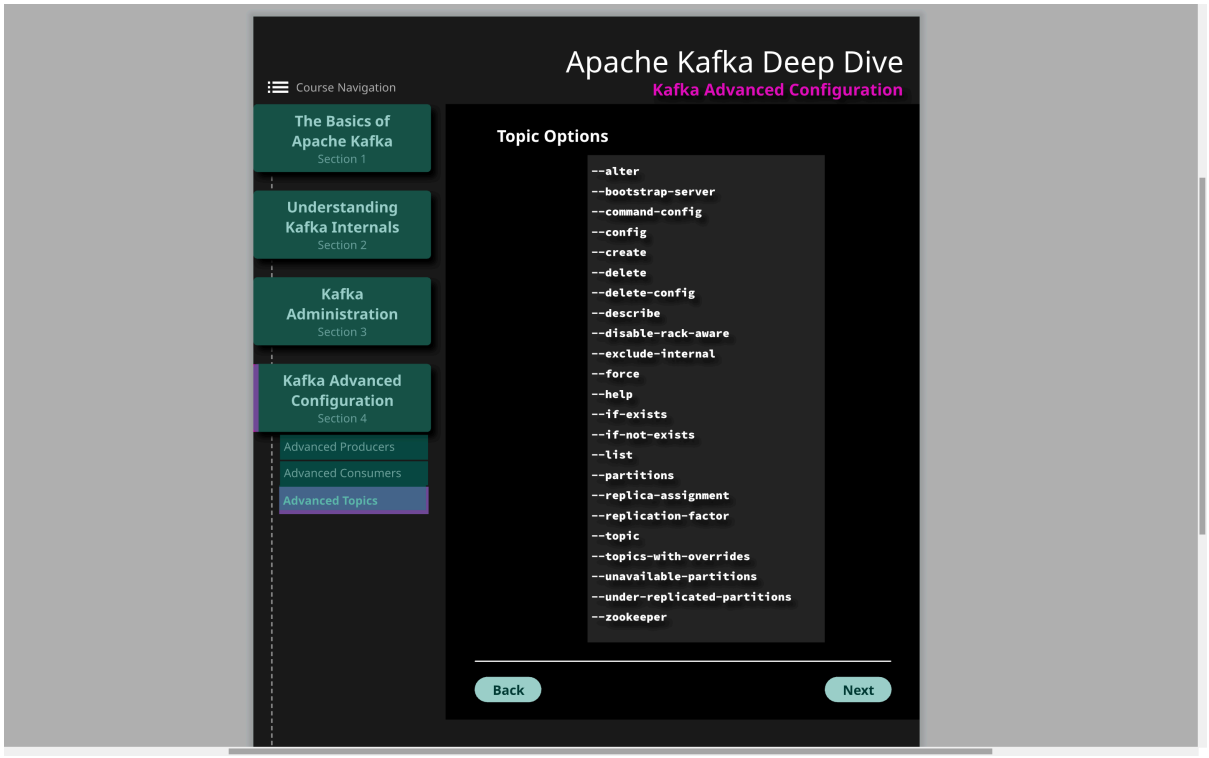
Topic Design

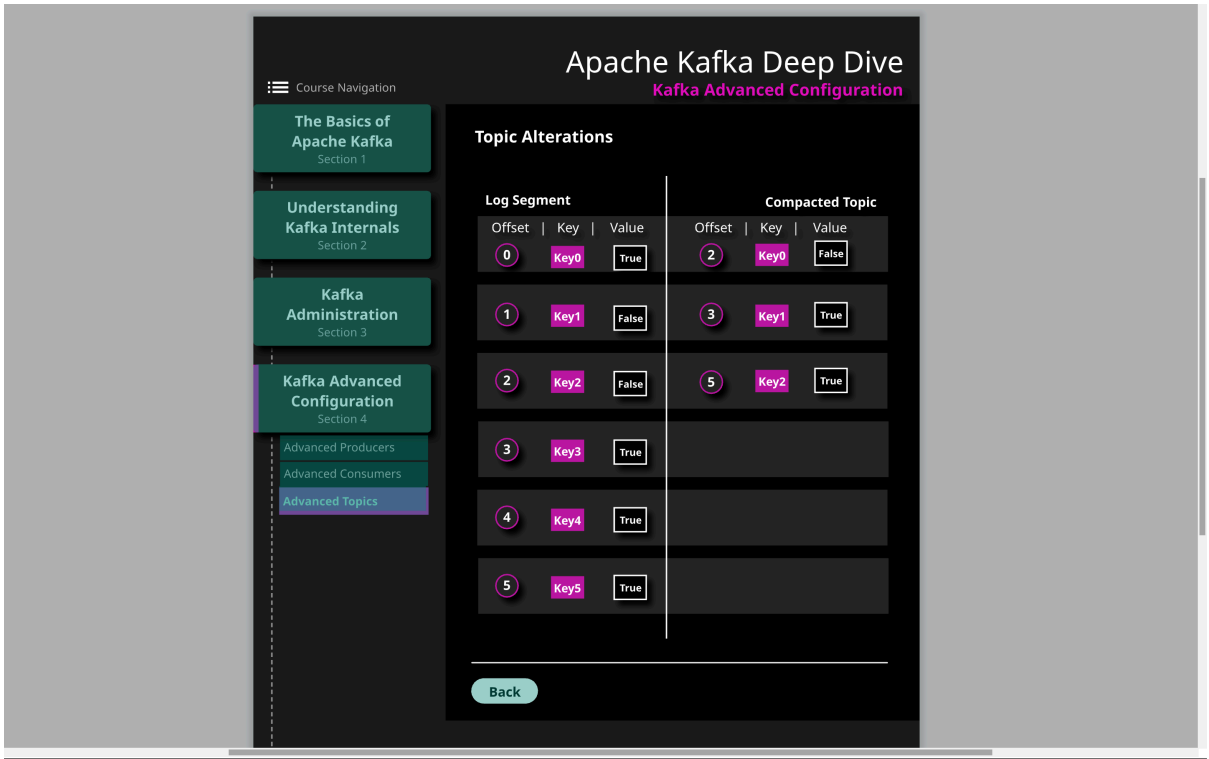


Design Considerations:

- Data accuracy
- Popularity of events
- Amount of data to process

Next


Course Navigation
Apache Kafka Deep Dive
Kafka Advanced Configuration
The Basics of Apache Kafka
Section 1
Understanding Kafka Internals
Section 2
Kafka Administration
Section 3
Kafka Advanced Configuration
Section 4
Advanced Producers
Advanced Consumers
Advanced Topics
Topic Options
--alter
--bootstrap-server
--command-config
--config
--create
--delete
--delete-config
--describe
--disable-rack-aware
--exclude-internal
--force
--help
--if-exists
--if-not-exists
--list
--partitions
--replica-assignment
--replication-factor
--topic
--topics-with-overrides
--unavailable-partitions
--under-replicated-partitions
--zookeeper
Back
Next


Course Navigation
Apache Kafka Deep Dive
Kafka Advanced Configuration
The Basics of Apache Kafka
Section 1
Understanding Kafka Internals
Section 2
Kafka Administration
Section 3
Kafka Advanced Configuration
Section 4
Advanced Producers
Advanced Consumers
Advanced Topics
Topic Alterations
Log Segment
Offset | Key | Value
0 Key0 True
1 Key1 False
2 Key2 False
3 Key3 True
4 Key4 True
5 Key5 True
Compacted Topic
Offset | Key | Value
2 Key0 False
3 Key1 True
5 Key2 True
Back