

A large, light gray play button icon is positioned on the left side of the slide. It consists of a white triangle pointing to the right, centered within a series of concentric circles that create a sense of depth and motion.

Kafka Topics, Partitions, Consumers

1. Topics and partitions
2. Partitions and replication
3. Consumers
4. Kafka commands

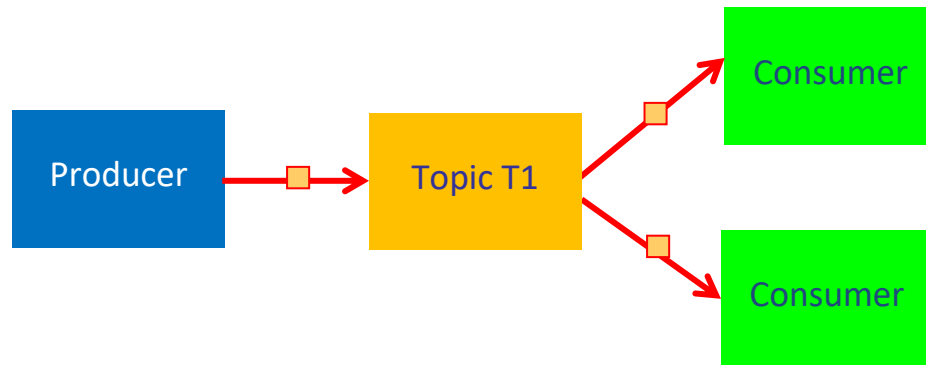


1. Topics and Partitions

- Recap of topics
- Topics are partitioned
- How partitioning works
- Defining a custom partitioner
- Partitioning strategies

Recap of Topics

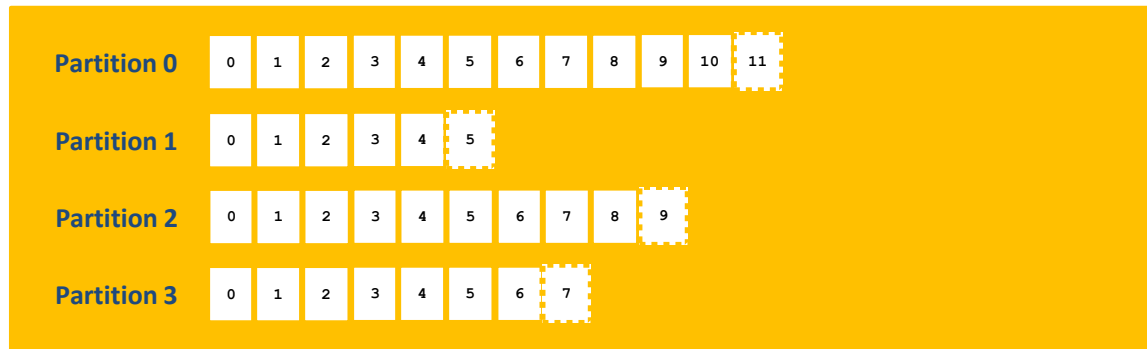
- A topic is a logical sequential collection of messages
 - A producer process publishes messages to a topic
 - Consumer processes pull messages from a topic



- Note:
 - The docs refers to messages as *records*, so we will too 😊

Topics are Partitioned

- In Kafka, each topic is partitioned
 - You specify the number of partitions when you create the topic



- Records are ordered within each partition
 - Writes to each partition are sequential
- Each record in a partition is assigned a sequential id
 - This is called the *offset*, identifies each record in a partition

How Partitioning Works

- When a producer sends a record, the record is a key/value pair
 - The key (optional) is a partition specifier
 - The value is the actual record data
- If Kafka receives a record that doesn't have a key:
 - Kafka use a round-robin partition strategy
- If Kafka receives a record that does have a key:
 - Kafka uses a default partitioner to choose the partition...
 - The default partitioner hashes the key...
 - Kafka uses the hash to determine the partition

Defining a Custom Partitioner

- You can also define a custom partitioner
 - A custom partitioner performs semantic processing on a key
 - E.g. if a key is a country, we can map it to a partition per continent
- You can implement a custom partitioner as a Java class
 - Kafka will pass records to the custom partitioner
 - The custom partitioner has access to a record's key/value
 - The custom partitioner decides the partition for the record
- We'll show how to do all this later

Partitioning Strategies

- Each partition is consumed by a designated consumer
 - We'll describe how this works shortly
- Random partitioning is recommended
 - When the producer sends a record, specify a random key
 - So records will be evenly distributed across partitions
 - So consumers will have an even distribution of work to do
- Aggregated partitioning is another possibility
 - Scenario: In an online retailer app, each customer has an ID
 - The producer can use the customer ID as a record key
 - All records for the same customer will go in the same partition, and will therefore be processed by the same consumer
 - This enables the consumer to aggregate data per customer

2. Partitions and Replication

- Overview
- Leader vs. follower brokers
- Additional technical info

Overview

- Replication is a critical ingredient in production systems
 - Data must be replicated, to avoid a single point of failure
- In Kafka, replication works at the partition level
 - The default replication factor is 3
 - i.e. each partition is stored on 3 separate brokers

Leader vs. Follower Brokers

- For each partition:
 - 1 broker is designated as the *leader* broker
 - The other brokers (e.g. 2 others) are designated as *followers*
- All reads/writes go to the leader broker
 - The follower brokers just fulfil a backup role
- E.g. when a producer publishes a record, it goes to the leader
 - The leader appends the record to its log and increments the offset
 - The leader propagates the record to the follower brokers
 - Acknowledgments are issued, to indicate successful delivery

Additional Technical Info

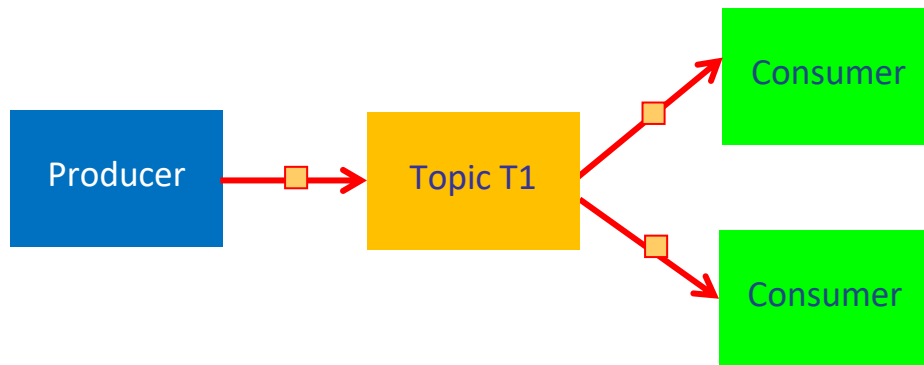
- How does a producer know which broker is the leader for a particular partition?
 - The producer contacts the cluster to determine partition leaders
- How does Kafka distribute leadership?
 - Fairly, i.e. each broker is a leader for a fair share of partitions
- What happens when a leader broker goes down?
 - Kafka promotes one of the followers to become the leader
- Where is data actually stored?
 - Each partition is stored as a separate directory on the file system
 - There are 2 files - the data (*.log file) and an index (*.index file)

3. Consumers

- Recap of consumers
- Consumer groups
- Consumers and partitions
- Best practices

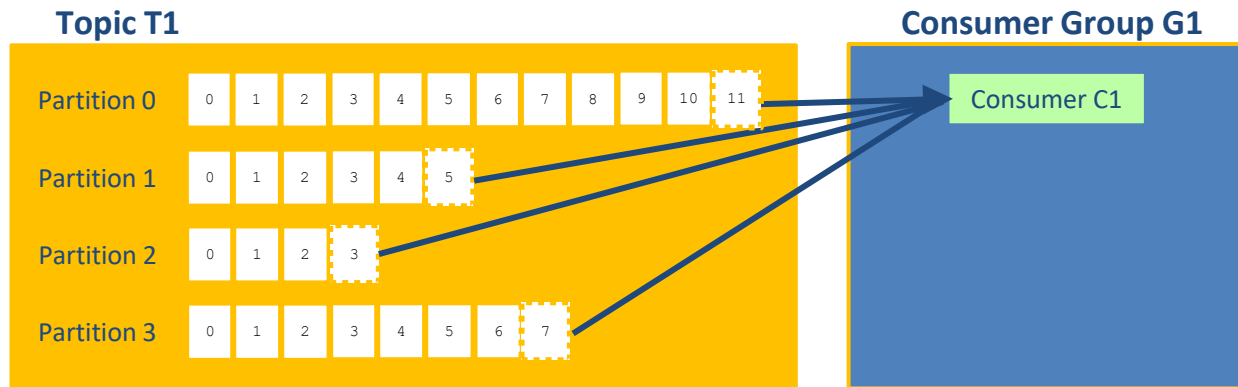
Recap of Consumers

- A consumer is a process that subscribes to a topic and reads records from that topic



Consumer Groups

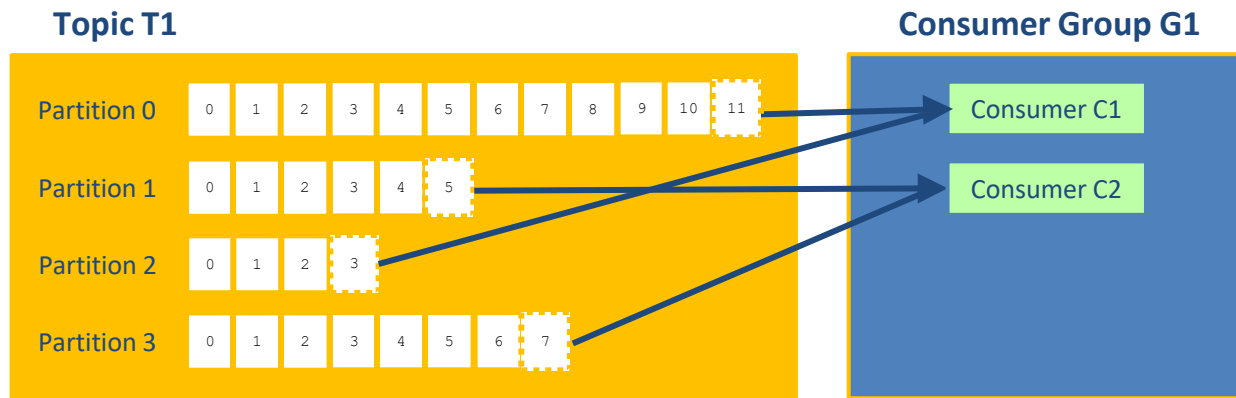
- Kafka consumers are all part of a consumer group
 - A consumer group can have multiple consumers



- Consider the scenario above:
 - Topic T1 has 4 partitions
 - Consumer group G1 has one consumer C1
 - If C1 subscribes to T1, it gets all records from all 4 partitions in T1

Consumers and Partitions (1 of 3)

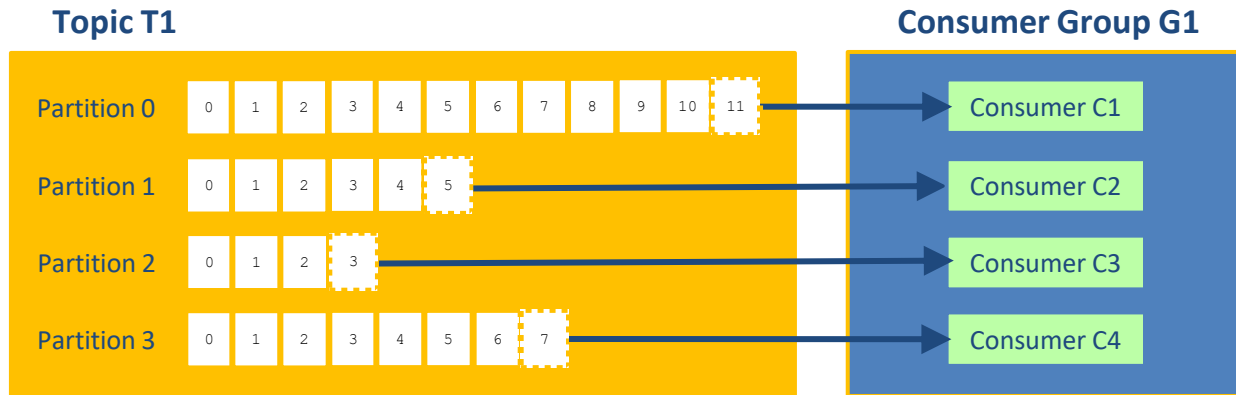
- If there are multiple consumers in a group...
 - Each consumer in the group will handle specific partition(s)
 - All records in a particular partition go to the same consumer



- Consider the scenario above:
 - Consumer group G1 has fewer consumers than there are partitions
 - Kafka will share the partitions evenly amongst the consumers

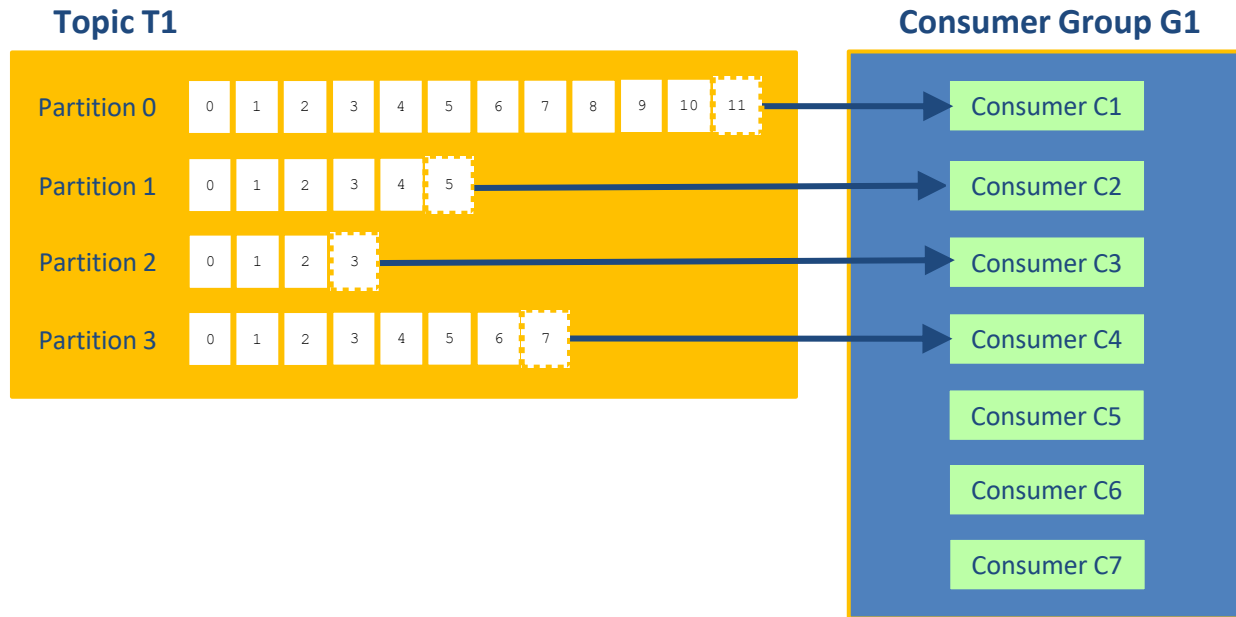
Consumers and Partitions (2 of 3)

- If there are the same number of consumers in a group as there are partitions in the topic...
 - Each consumer in the group will handle a specific partition



Consumers and Partitions (3 of 3)

- If there are more consumers in a group than there are partitions in the topic...
 - Surplus consumers are idle - they never receive any records



Best Practices

- The primary way to improve data consumption is to add more consumers to a consumer group
 - Consumers might be doing time-intensive work
 - A single consumer can't keep up with the rate of data as it arrives
 - Having multiple consumers helps achieve parallelization
- This is a good reason to create topics with lot of partitions
 - The more partitions, the more dedicated consumers you can have
 - You can scale up the number of consumers, based on the load
- Note:
 - Kafka rebalances partitions when a new consumer joins a group
 - Consumers are reassigned a new bunch of partitions to handle

4. Kafka Commands

- Overview
- Creating a topic
- Describing topics
- Deleting a topic
- Publishing records to a topic
- Consuming records from a topic

Overview

- Kafka provides various utilities you can run from a shell or Windows command prompt
 - Create / delete topics
 - Describe topics
 - Publish records to a topic
 - Consume records from a topic
 - Etc.
- To run these commands, open a shell / command window and go to the following directory
 - In Linux/OSX: Kafka `bin` directory
 - In Windows: Kafka `bin/windows` directory

Creating a Topic (1 of 2)

- To create a topic (e.g. in a Windows command prompt):

```
kafka-topics --create ^  
              --bootstrap-server localhost:9092 ^  
              --replication-factor 1 ^  
              --partitions 1 ^  
              --topic my-topic-a
```

- Note about `kafka-topics` options:
 - `bootstrap-server` can be any node in the cluster
 - `replication-factor` default is 1 (in production, 3 is good)
 - `partitions` default is 1 (in production, specify a suitable value)
- Aside: Line continuations characters for different shells:
 - `^` Windows command prompt
 - `\` Linux or macOS shell window
 - ``` Windows PowerShell window

Creating a Topic (2 of 2)

- To change the number of partitions in a topic:

```
kafka-topics --alter ^  
--bootstrap-server localhost:9092 ^  
--partitions 3 ^  
--topic my-topic-a
```

- To create a topic with multiple partitions in the first place:

```
kafka-topics --create ^  
--bootstrap-server localhost:9092 ^  
--replication-factor 1 ^  
--partitions 5 ^  
--topic my-topic-b
```

Describing Topics

- To describe topics:

```
kafka-topics --bootstrap-server localhost:9092 --describe
```



```
C:\Windows\System32\cmd.exe

C:\Apps\kafka_2.12-2.4.1\bin\windows>kafka-topics --bootstrap-server localhost:9092 --describe
Topic: my-topic-a      PartitionCount: 3      ReplicationFactor: 1    Configs: segment.bytes=1073741824
  Topic: my-topic-a      Partition: 0      Leader: 0      Replicas: 0      Isr: 0
  Topic: my-topic-a      Partition: 1      Leader: 0      Replicas: 0      Isr: 0
  Topic: my-topic-a      Partition: 2      Leader: 0      Replicas: 0      Isr: 0
Topic: my-topic-b      PartitionCount: 5      ReplicationFactor: 1    Configs: segment.bytes=1073741824
  Topic: my-topic-b      Partition: 0      Leader: 0      Replicas: 0      Isr: 0
  Topic: my-topic-b      Partition: 1      Leader: 0      Replicas: 0      Isr: 0
  Topic: my-topic-b      Partition: 2      Leader: 0      Replicas: 0      Isr: 0
  Topic: my-topic-b      Partition: 3      Leader: 0      Replicas: 0      Isr: 0
  Topic: my-topic-b      Partition: 4      Leader: 0      Replicas: 0      Isr: 0

C:\Apps\kafka_2.12-2.4.1\bin\windows>_
```

↑
Isr means
"in-sync replicas"

Deleting a Topic

- To delete a topic (and all its partitions):

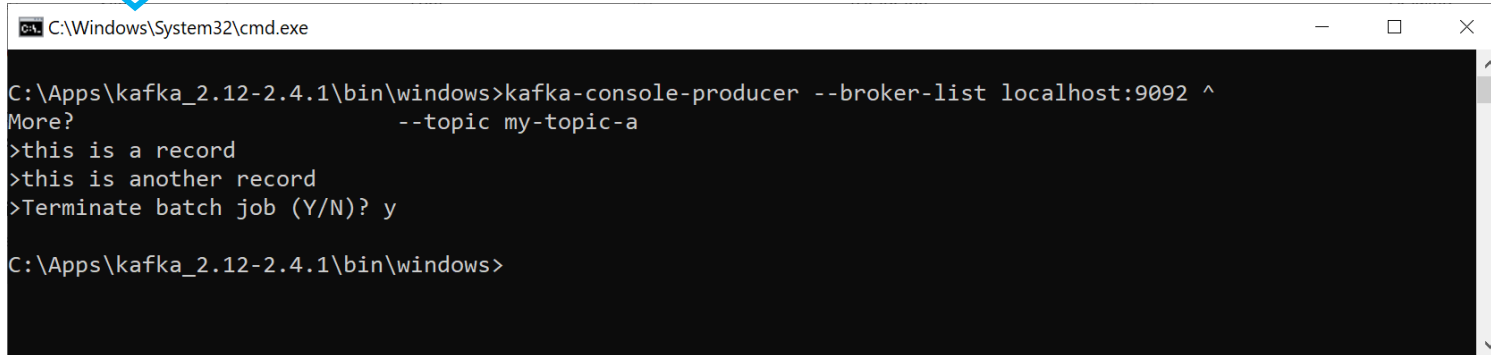
```
kafka-topics --delete ^  
--bootstrap-server localhost:9092 ^  
--topic my-topic-b
```

- NOTE: This might cause an exception, based on file system permissions!

Publishing Records to a Topic (1 of 2)

- To publish records from the console to a topic:

```
kafka-console-producer --broker-list localhost:9092 ^  
--topic my-topic-a
```



```
C:\Windows\System32\cmd.exe  
  
C:\Apps\kafka_2.12-2.4.1\bin\windows>kafka-console-producer --broker-list localhost:9092 ^  
More? --topic my-topic-a  
>this is a record  
>this is another record  
>Terminate batch job (Y/N)? y  
  
C:\Apps\kafka_2.12-2.4.1\bin\windows>
```

- In this example, we've just specified record values
 - We haven't specified record keys
 - So Kafka will round-robin the records to partitions

Publishing Records to a Topic (2 of 2)

- To publish a record (key + value) to a topic:

```
kafka-console-producer --broker-list localhost:9092 ^  
--topic my-topic-a ^  
--property "parse.key=true" ^  
--property "key.separator=:"
```



```
C:\Windows\System32\cmd.exe  
  
C:\Apps\kafka_2.12-2.4.1\bin\windows>kafka-console-producer --broker-list localhost:9092 ^  
More? --topic my-topic-a ^  
More? --property "parse.key=true" ^  
More? --property "key.separator=:"  
>key1:This is a value with key1  
>key2:This is a different value with key2  
>Terminate batch job (Y/N)? y  
  
C:\Apps\kafka_2.12-2.4.1\bin\windows>
```

Consuming Records from a Topic (1 of 2)

- To consume records from a topic to the console:

```
kafka-console-consumer ^  
--bootstrap-server localhost:9092 ^  
--topic my-topic-a ^  
--from-beginning ^  
--property print.key=true ^  
--property print.value=true ^  
--key-deserializer "org.apache.kafka.common.serialization.StringDeserializer" ^  
--value-deserializer "org.apache.kafka.common.serialization.StringDeserializer"
```



```
C:\Windows\system32\cmd.exe  
  
C:\Apps\kafka_2.12-2.4.1\bin\windows>kafka-console-consumer ^  
More? --bootstrap-server localhost:9092 ^  
More? --topic my-topic-a ^  
More? --from-beginning ^  
More? --property print.key=true ^  
More? --property print.value=true ^  
More? --key-deserializer "org.apache.kafka.common.serialization.StringDeserializer" ^  
More? --value-deserializer "org.apache.kafka.common.serialization.StringDeserializer"  
null    this is a record  
null    this is another record  
key1    This is a value with key1  
key2    This is a different value with key2  
Processed a total of 4 messages  
Terminate batch job (Y/N)? y  
  
C:\Apps\kafka_2.12-2.4.1\bin\windows>
```

Consuming Records from a Topic (2 of 2)

- A consumer is in control over which records it consumes, e.g. `kafka-console-consumer` has these options:
 - `--from-beginning` - get all records from beginning
 - `--offset` - get records from offset for a partition
 - `--max-messages` - max number of records, then exit
- Example:

```
kafka-console-consumer ^
--bootstrap-server localhost:9092 ^
--topic my-topic-a ^
--partition 2 ^
--offset 1 ^
--max-messages 2 ^
--property print.key=true ^
--property print.value=true ^
--key-deserializer "org.apache.kafka.common.serialization.StringDeserializer" ^
--value-deserializer "org.apache.kafka.common.serialization.StringDeserializer"
```

A large, light gray play button icon is positioned on the left side of the slide. It consists of a white right-pointing triangle centered within a series of concentric circles in varying shades of gray.

Summary

- Topics and partitions
- Partitions and replication
- Consumers
- Kafka commands

