Big Data Hadoop and Spark Developer

Data Ingestion into Big Data Systems and ETL

Learning Objectives

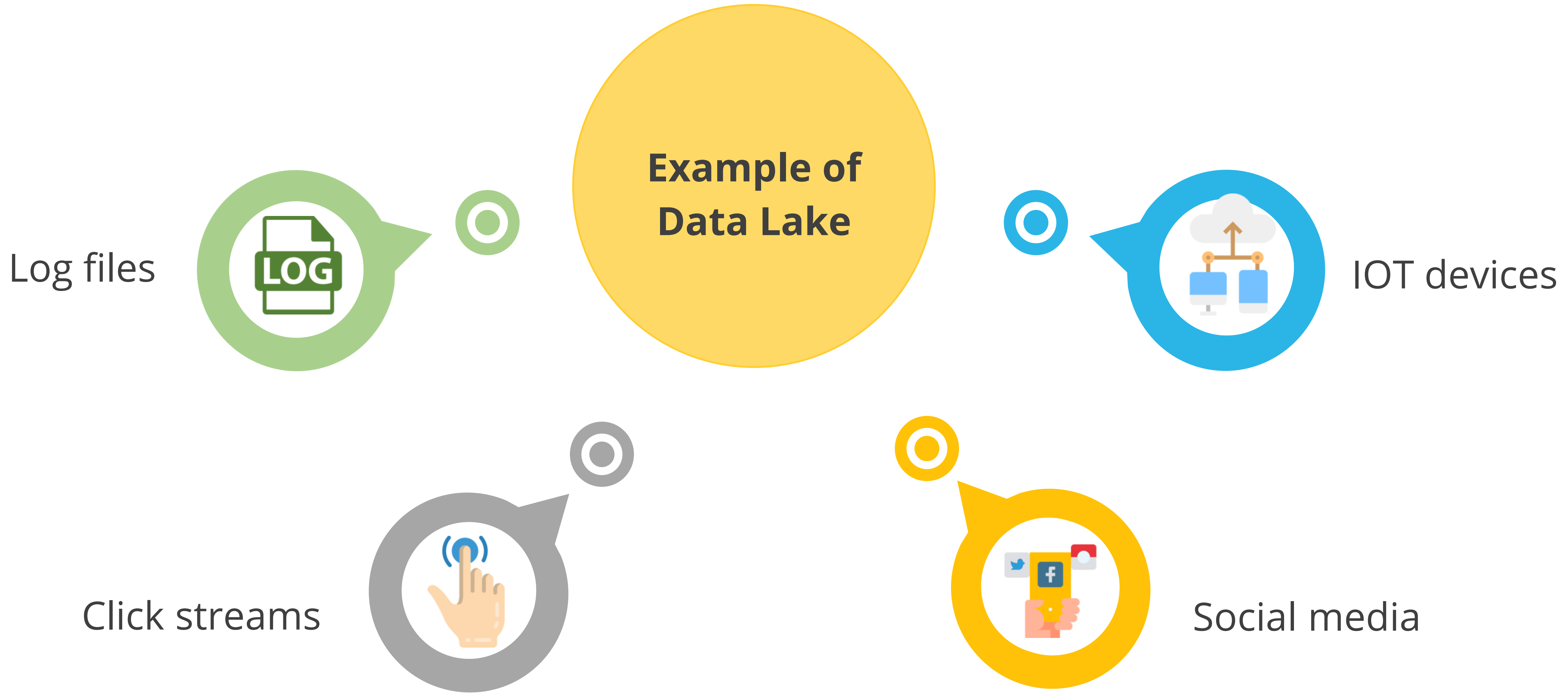
By the end of this lesson, you will be able to:

- Identify various Big data ingestion tools
- Summarize Apache Kafka architecture, producer, consumer, and cluster
- Demonstrate the capabilities and the applications of Apache Flume
- Study the architecture and components of Apache Flume

Data Ingestion Overview

Data Lake

A data lake is a centralized storage repository where enormous volumes of organized, semi-structured, and unstructured data can be stored.



Data Lake vs. Data Warehouse

The following table depicts the difference between a data lake and a data warehouse:

Characteristics	Data Lake	Data Warehouse
Data structure	Raw	Processed
Purpose of data	Not determined	Currently in use
Price and performance	Low-cost storage	Expensive storage
Users	Data scientists and data developers	Business analysts
Type of analytics	Machine learning, predictive analytics, and data discovery	Batch reporting, BI, and visualizations

Data Ingestion

01

Big data ingestion is the process of moving data, particularly unstructured data, from its source into a system, such as Hadoop, for storage and analysis.

02

The ingestion process may be continuous or asynchronous, real-time or batch, or both, depending on the characteristics of the source and destination.

03

Data transformation or conversion makes the data accessible to the target system when the source and destination systems do not share the same data format or protocol.

Big Data Ingestion Tools

It is necessary to select the right data ingestion tool, which is based on factors such as data source, goal, and transformations.

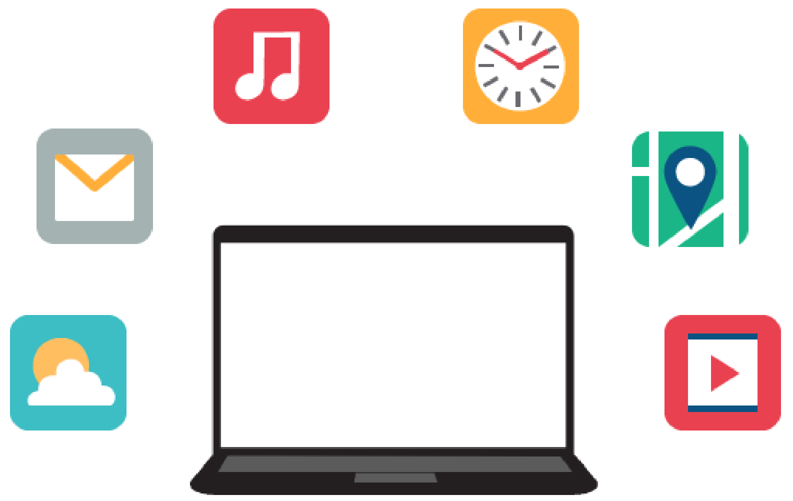
Apache Flume

Apache Kafka

Big Data Ingestion Tools

Big Data Ingestion Tools

- Data ingestion solutions give users a framework for extracting data from many sources and support a variety of data transport protocols.
- Data ingestion technologies reduce the need to manually construct unique data pipelines for each data source, which effectively transfers data to ETL tools and speeds up processing.



Apache Kafka

Apache Kafka

- Apache Kafka is a real-time communication system with great performance.
- It is a free and open-source program that is part of the Apache project.
- Apache Kafka is mostly used to create real-time streaming data pipelines and applications that react to data streams.
- It combines communications, storage, and stream processing to allow both historical and real-time data to be stored and analyzed.

Apache Kafka: Characteristics

Apache Kafka has the following characteristics:

Kafka

- It can process and send millions of messages per second to many recipients.
- It is a partitioned and distributed messaging system.
- Kafka can readily handle large volumes of data streams.
- It has a high level of fault tolerance.

Apache Kafka: Use Case

In an organization, Kafka may be employed for a variety of purposes, including:

Messaging service	Sends and receives millions of messages in real-time
Real-time stream processing	Processes a continuous stream of information in real-time and pass it to stream processing systems, such as Storm
Log aggregation	Accepts physical log files from many systems and stores them in a centralized place, such as HDFS

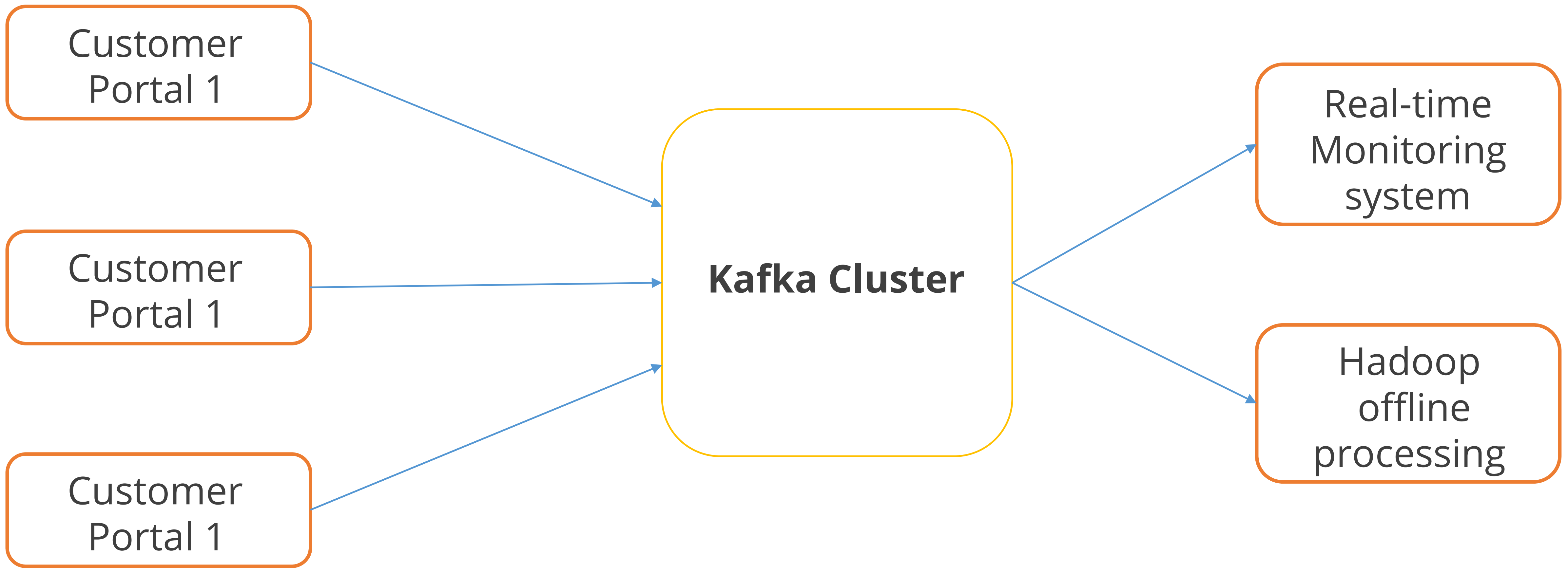
Apache Kafka: Use Case

In an organization, Kafka may be employed for a variety of purposes, including:

Commit log service	Can be used as an external commit log for distributed systems
Event sourcing	Maintains a time ordered sequence of events
Website activity tracking	Processes real-time website activity that users can take, such as page views, searches, or other actions

Apache Kafka: Aggregating User Activity

Apache Kafka can aggregate user activity data, such as clicks, navigation, and searches, from different websites of an organization. Such user activities can be sent to a real-time monitoring system and Hadoop system for offline processing.



Apache Kafka: LinkedIn

LinkedIn uses Apache Kafka manage streams of information.

Monitoring

- This method collects the metrics.
- This technology created the dashboards for monitoring.

Messaging

- Message queues are used in content feeds.
- It is used as a publish-subscribe system for searches and content feeds.

Apache Kafka: LinkedIn

LinkedIn uses Apache Kafka manage streams of information.

Analytics

- Its purpose is to collect page visits and clicks.
- It is archived in a centralized Hadoop-based analytics system.

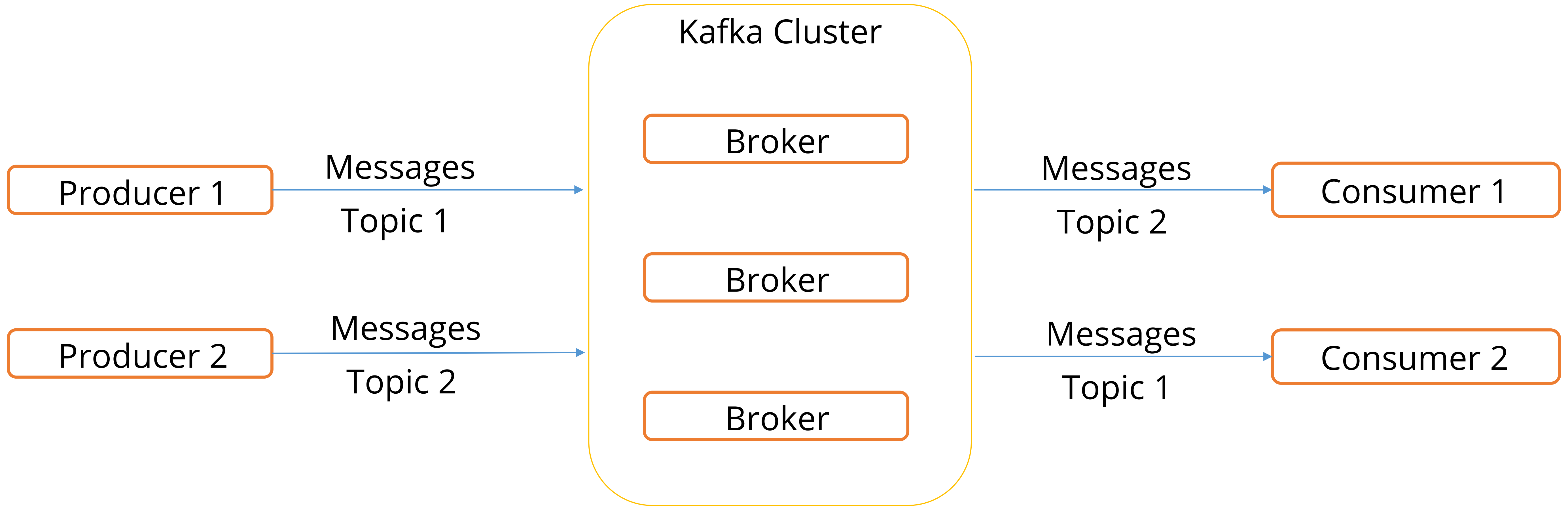
A building block for distributed applications

- It is designed for usage with distributed databases.
- It can be used in conjunction with distributed log systems.

Kafka Data Model

Kafka Data Model

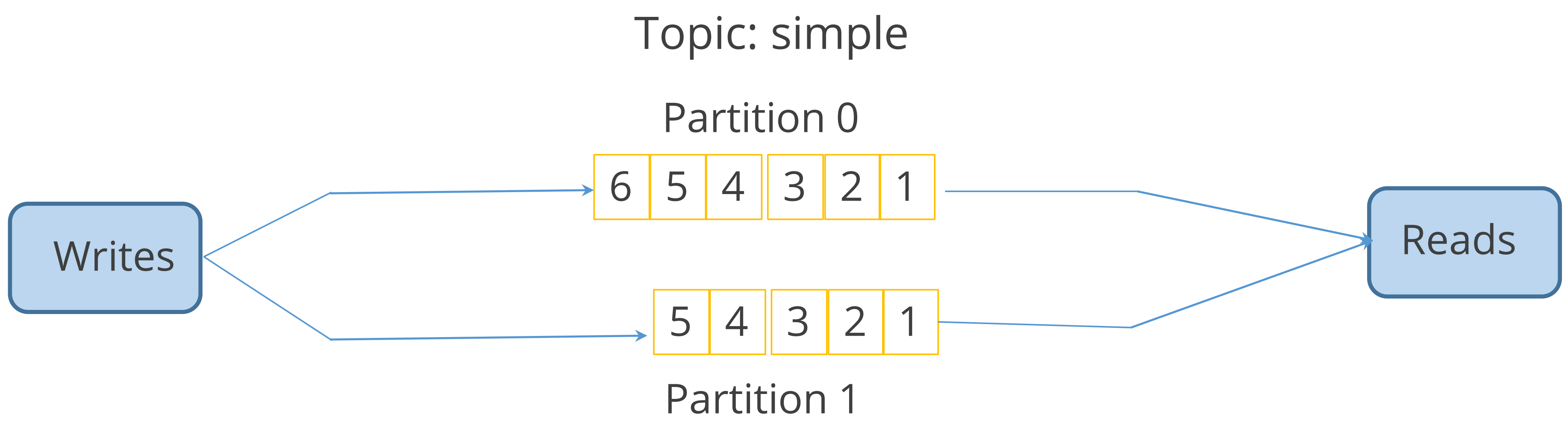
The Kafka data model consists of messages and topics.



- Messages represent information, such as lines in a log file, a row of stock market data, or an error message.
- Messages are grouped into categories called **topics**, such as Log Message and Stock Message.

Kafka Data Model: Topics

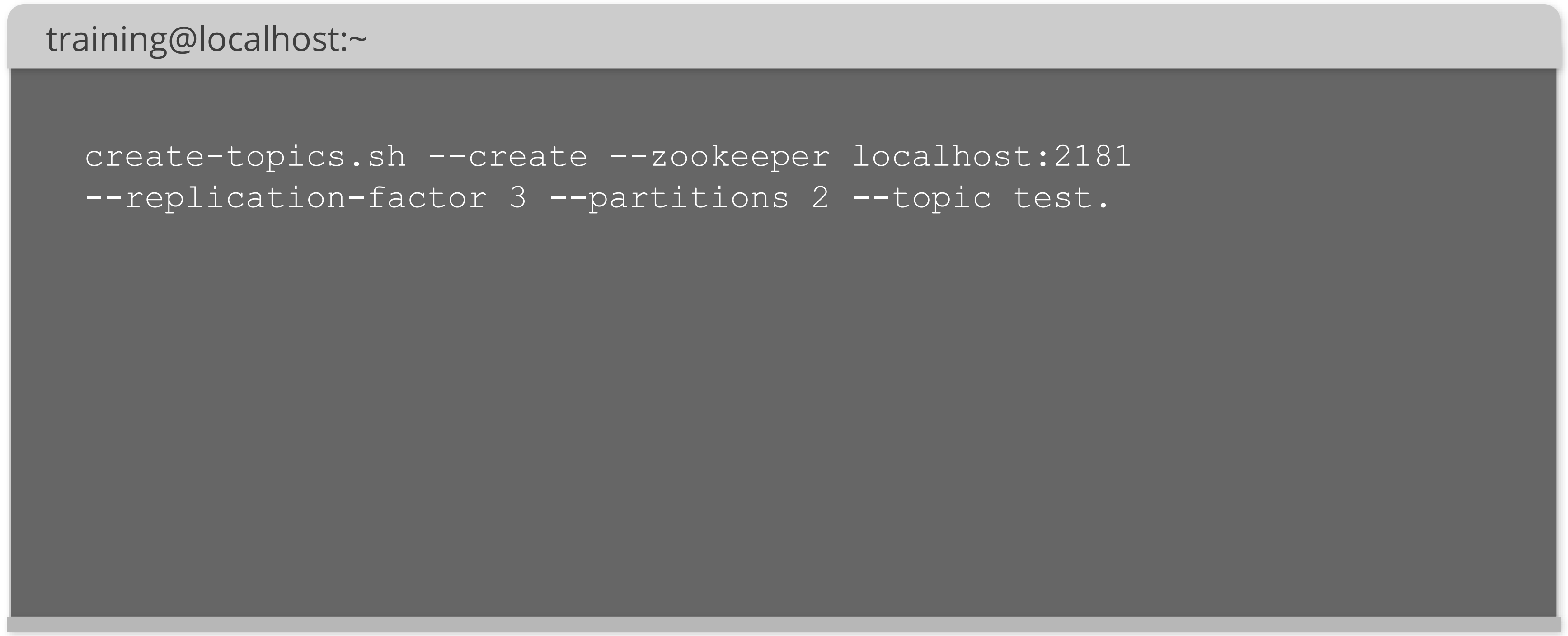
A topic is a category of messages in Kafka.



A topic is divided into one or more partitions which consist of the ordered set of messages.

Kafka Data Model: Topics

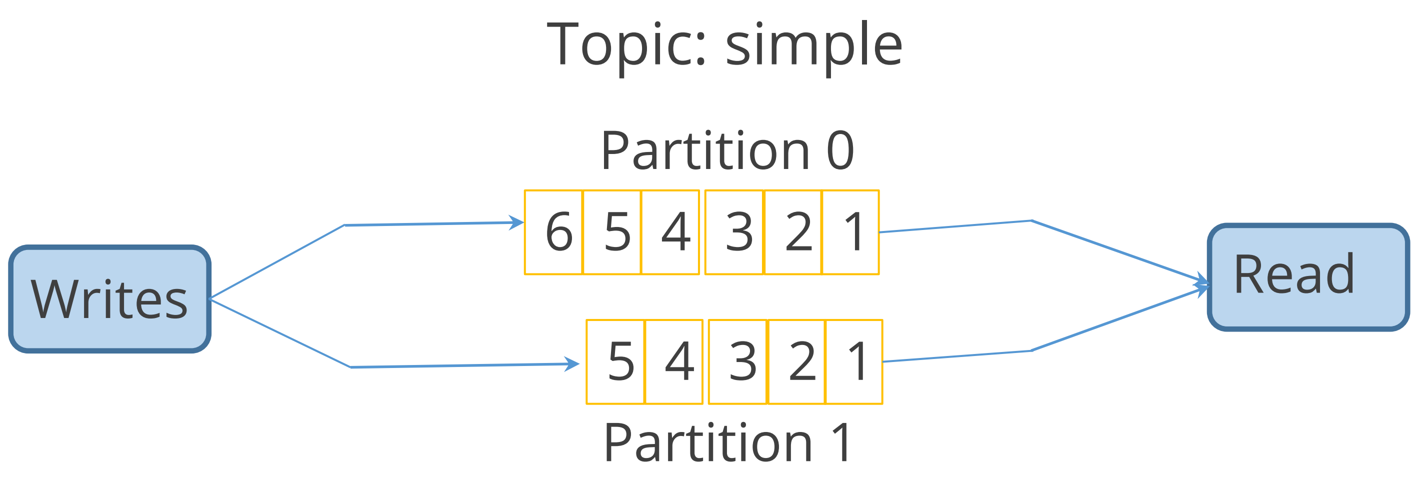
Kafka provides the kafka-topics.sh command to create and modify topics.


training@localhost:~

```
create-topics.sh --create --zookeeper localhost:2181
--replication-factor 3 --partitions 2 --topic test.
```

Kafka Data Model: Partitions

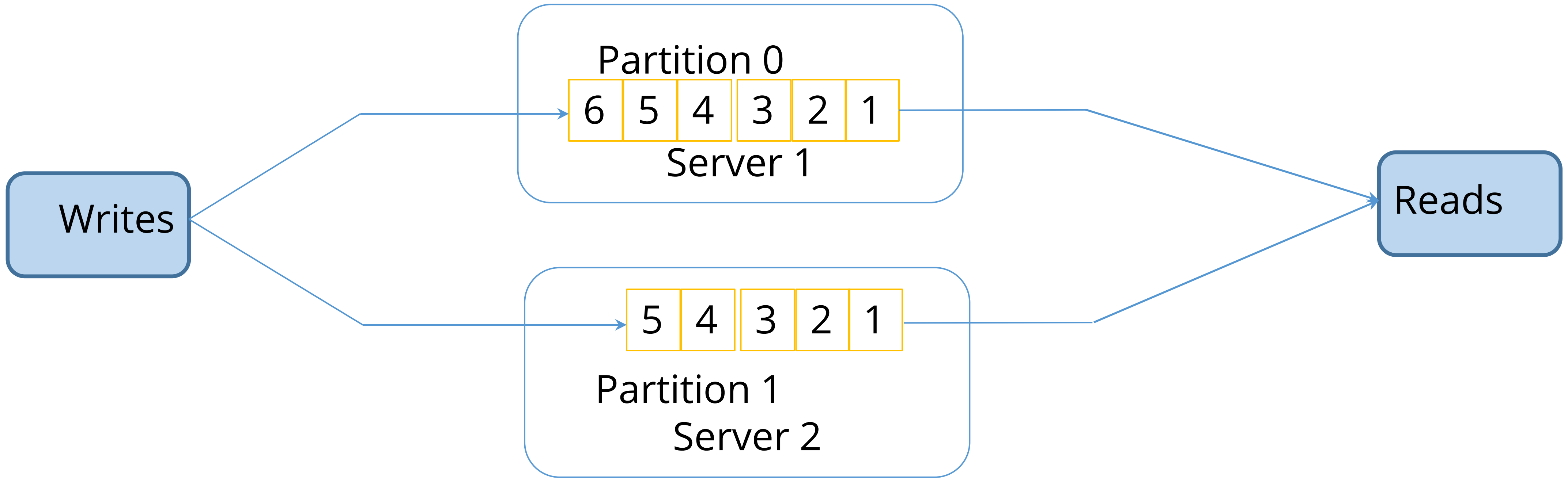
Topics are divided into partitions, which are the unit of parallelism in Kafka.



- Partitions allow messages in a topic to be distributed to multiple servers.
- A topic can have any number of partitions.
- Each partition should fit in a single Kafka server.
- The number of partitions decides the parallelism of the topic.

Kafka Data Model: Partition Distribution

Partitions can be distributed across the Kafka cluster.

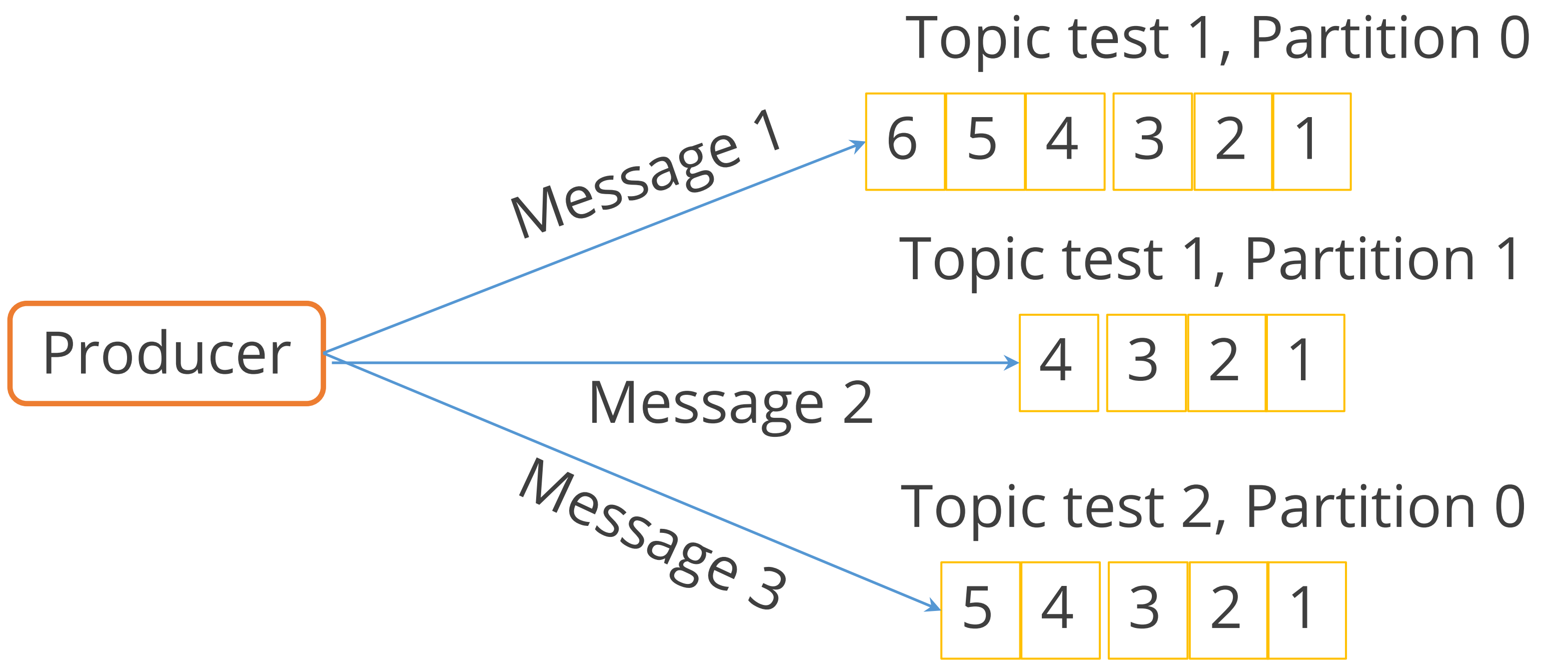


Kafka Data Model: Partition Distribution

- Each Kafka server may handle one or more partitions.
- One server is marked as a leader for the partition and the others are marked as followers.
- The leader controls the read and writes for the partition, whereas the followers replicate the data.
- If a leader fails, one of the followers automatically becomes the leader.
- ZooKeeper is used for the leader selection.

Kafka Data Model: Producers

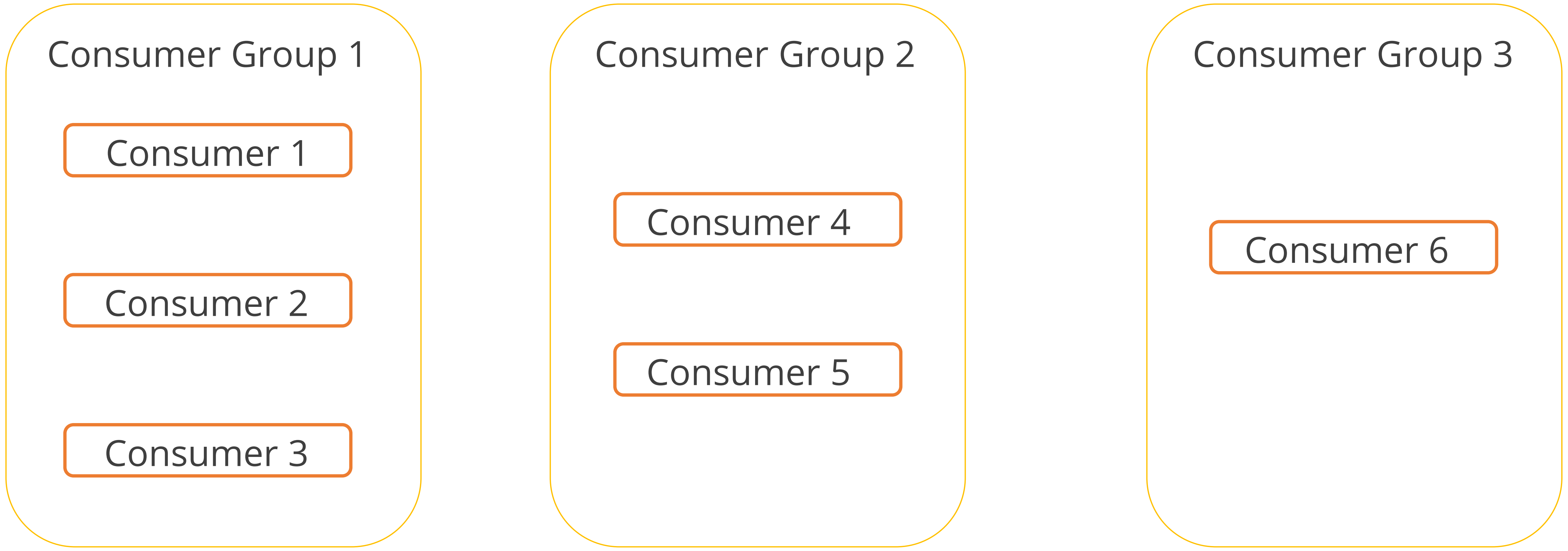
The producer is the creator of the message in Kafka.


Producer
Message 1
Message 2
Message 3
Topic test 1, Partition 0
6 5 4 3 2 1
Topic test 1, Partition 1
4 3 2 1
Topic test 2, Partition 0
5 4 3 2 1

- The producers place the message on a particular topic.
- The producers also decide which partition to place the message into.
- Topics should already exist before the producer places a message.
- Messages are added at one end of the partition.

Kafka Data Model: Consumers

The consumer is the receiver of the message in Kafka.



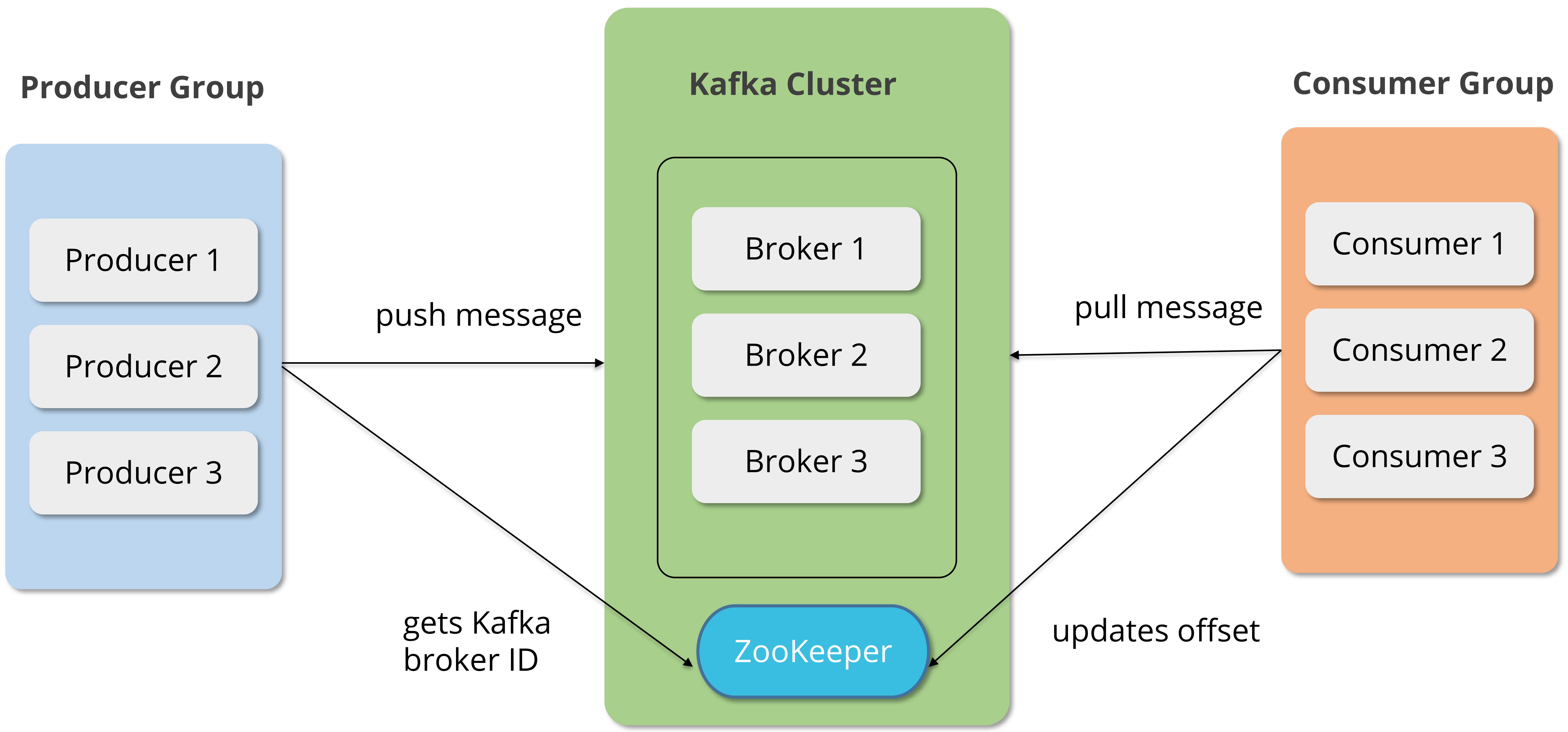
Kafka Data Model: Consumers

- Each consumer belongs to a consumer group.
- A consumer group may have one or more consumers.
- The consumers specify what topics they want to listen to.
- A message is sent to all the consumers in a consumer group.
- The consumer groups control the messaging system.

Apache Kafka Architecture

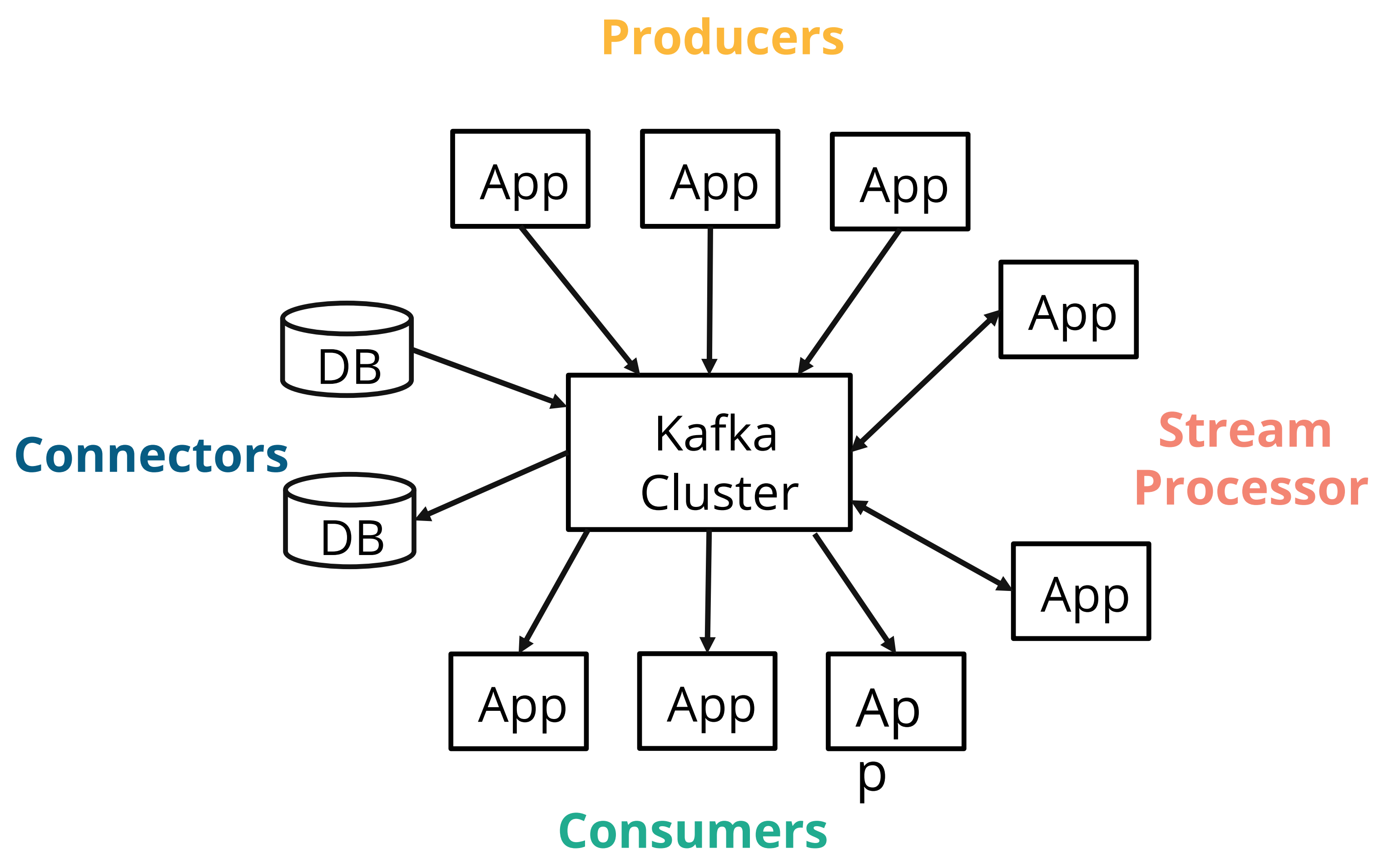
Kafka Architecture

The Kafka cluster architecture is depicted below.



Kafka APIs

Kafka has four core APIs:

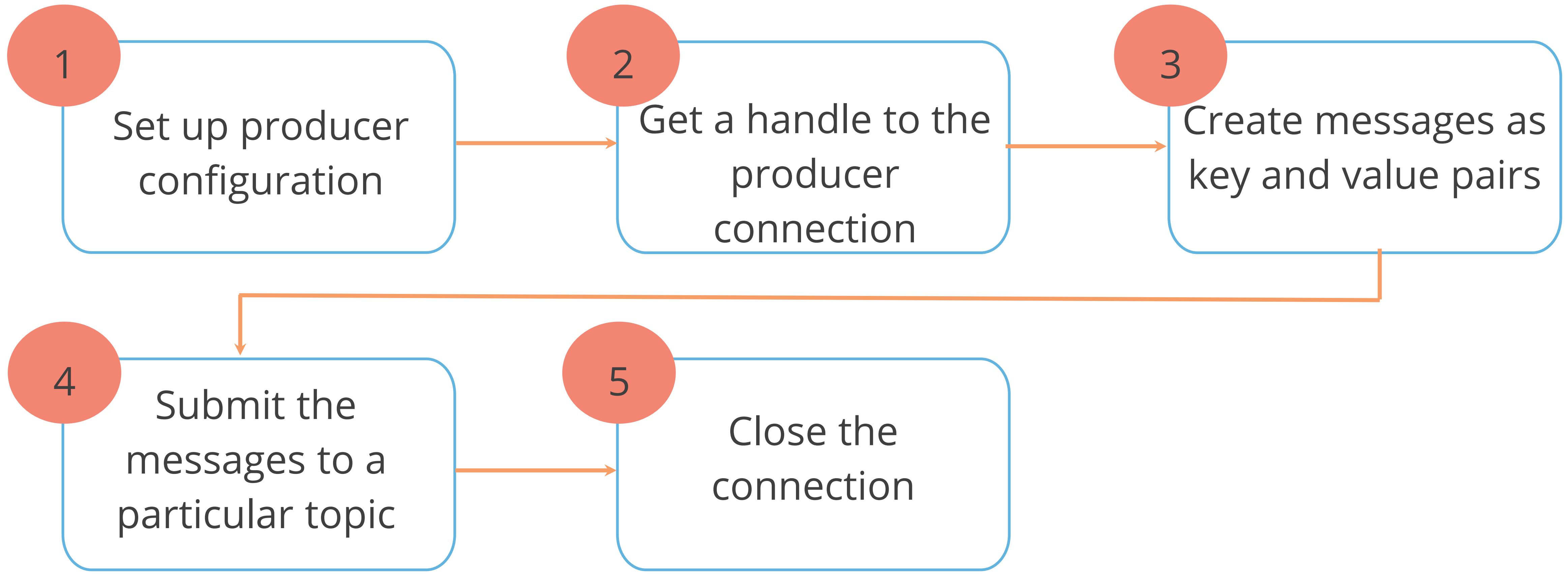


- The **Producer API** allows applications to send streams of data to topics in the Kafka cluster.
- The **Streams API** allows transforming the stream of data from input topics to output topics.
- The **Consumer API** allows applications to read streams of data from topics in the Kafka cluster.
- The **Connect API** allows data ingestion from some source system into Kafka or pushes from Kafka into some sink system.

Producer Side API

The producer side APIs provide an interface to connect to the cluster and insert messages into a topic.

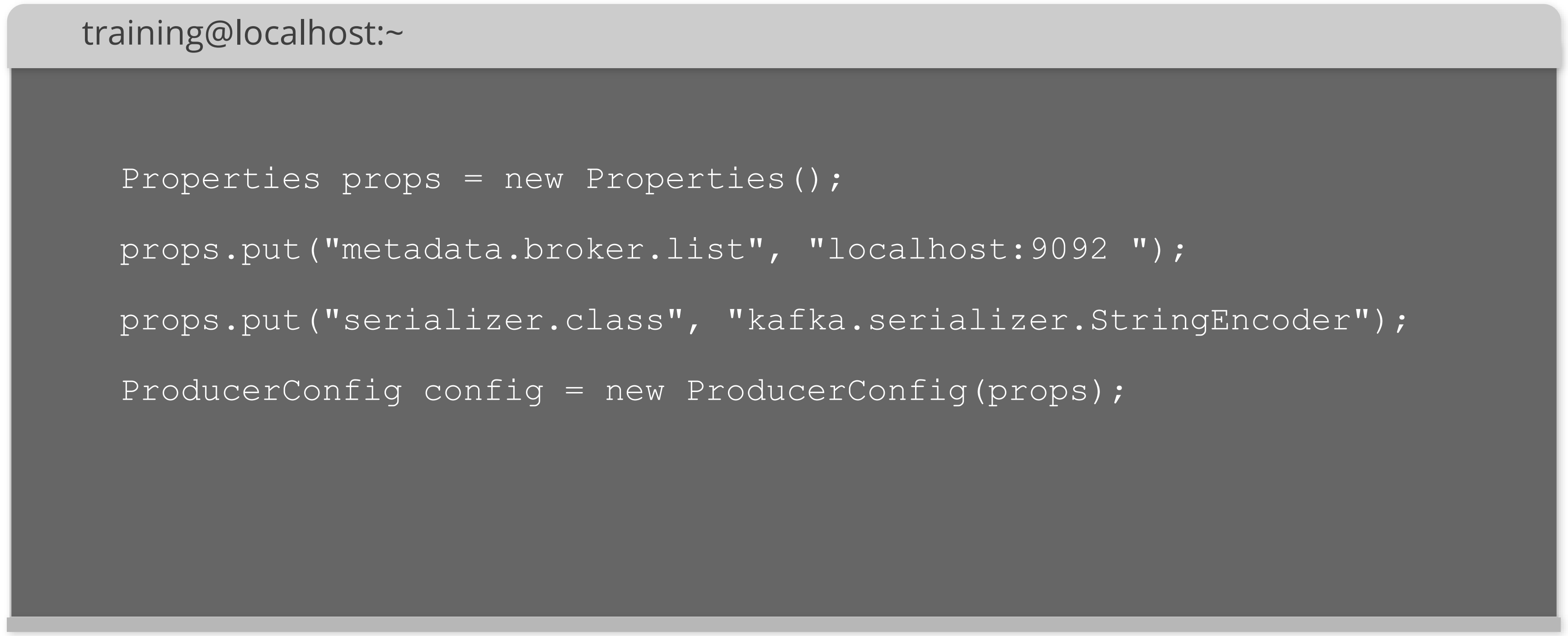
The steps involved in programming are as follows:


1
Set up producer configuration
2
Get a handle to the producer connection
3
Create messages as key and value pairs
4
Submit the messages to a particular topic
5
Close the connection

Producer Side API: Example

The producer transmits data to a Kafka topic by communicating with the Kafka broker hosts (worker nodes).

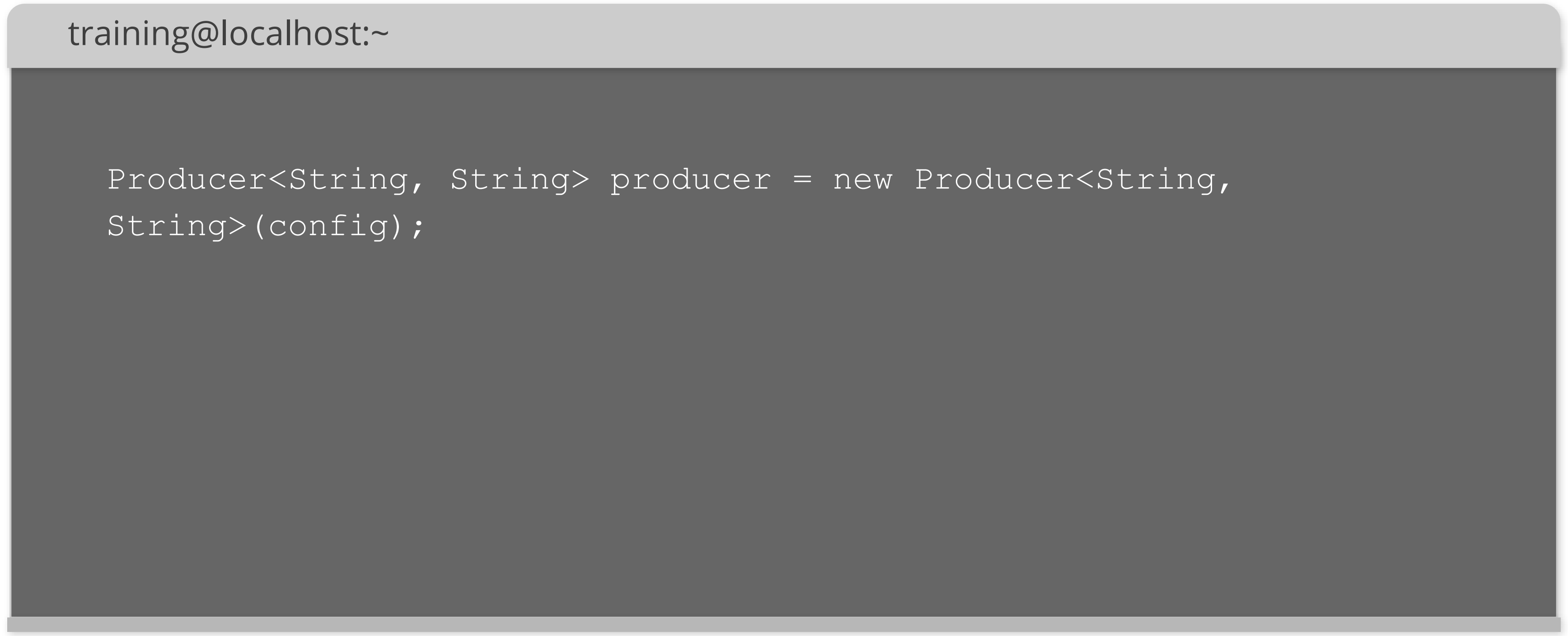
Step 1: Set up producer configuration



```
Properties props = new Properties();
props.put("metadata.broker.list", "localhost:9092 ");
props.put("serializer.class", "kafka.serializer.StringEncoder");
ProducerConfig config = new ProducerConfig(props);
```

Producer Side API: Example

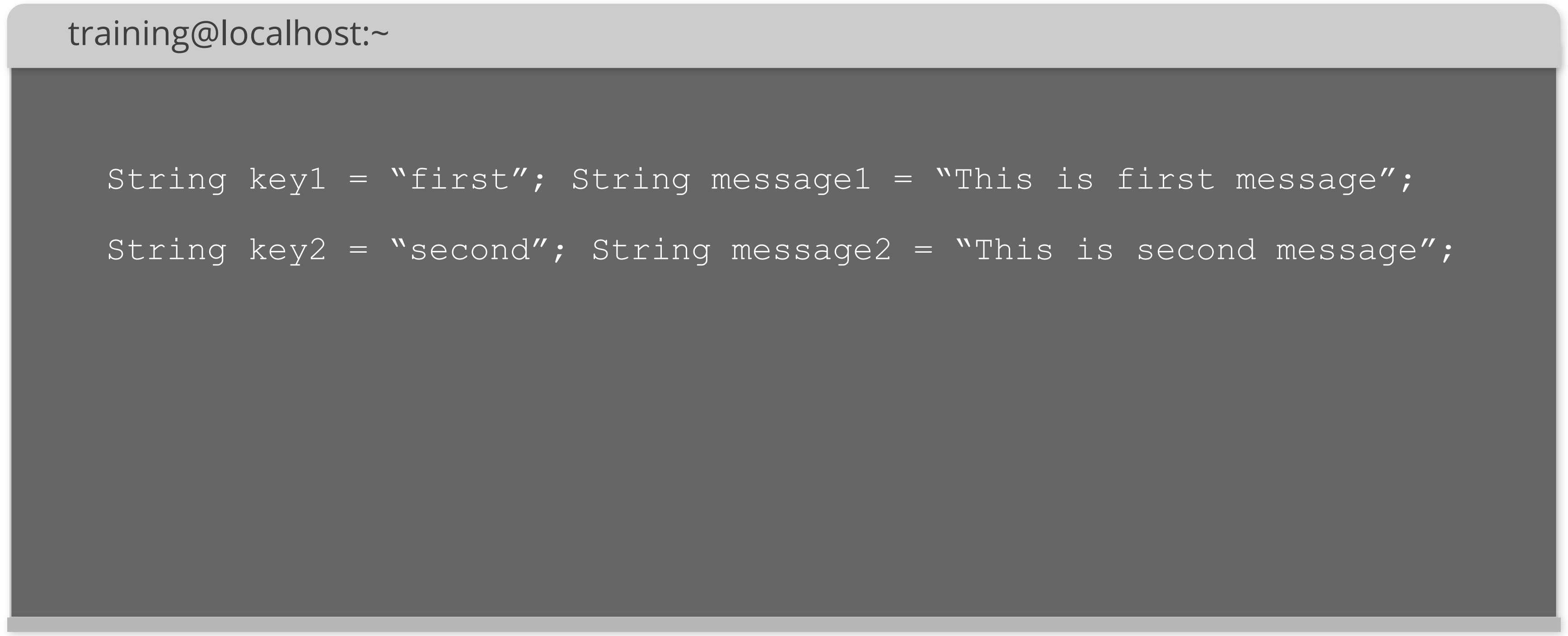
Step 2: Get a handle on the producer connection



```
Producer<String, String> producer = new Producer<String,
String>(config);
```

Producer Side API: Example

Step 3: Create messages as key and value pairs



training@localhost:~

```
String key1 = “first”; String message1 = “This is first message”;
String key2 = “second”; String message2 = “This is second message”;
```

Producer Side API: Example

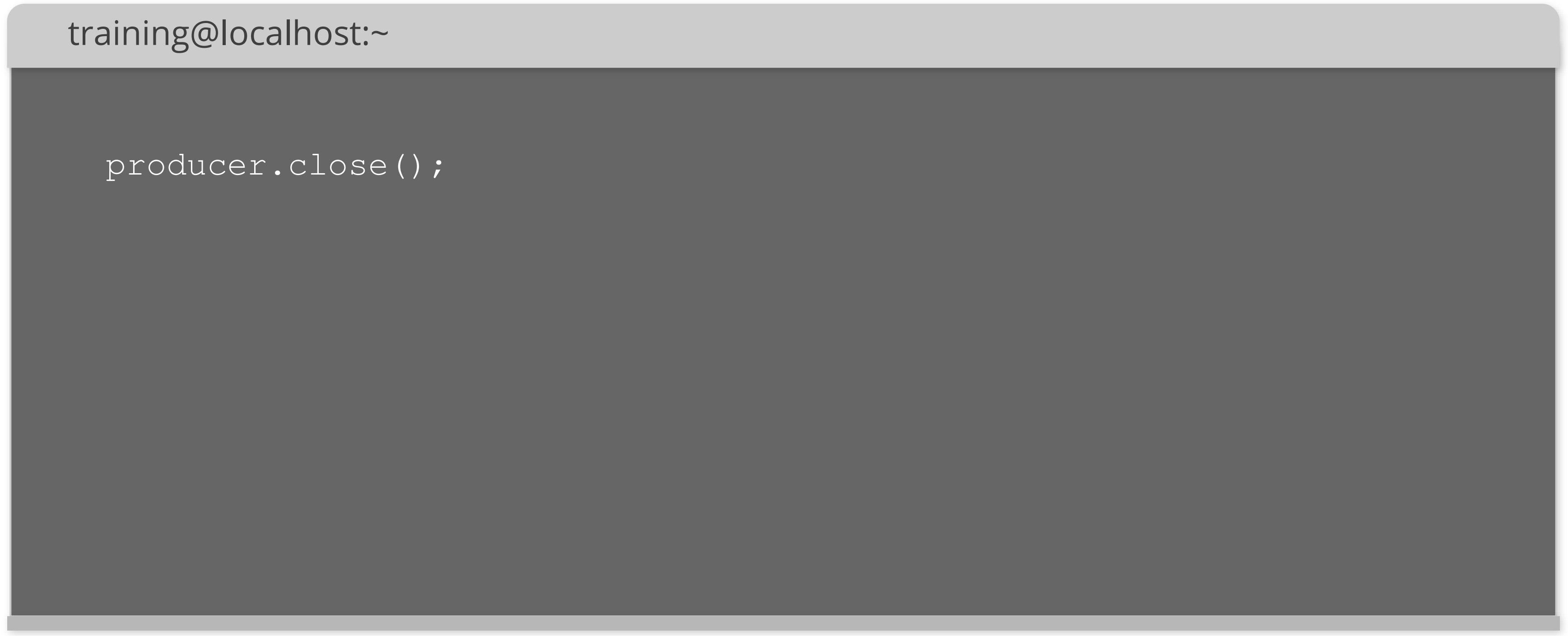
Step 4: Submit the messages to a particular topic

training@localhost:~

```
String topic = “test”;

KeyedMessage<String, String> data = new KeyedMessage<String,
String>(topic, key1, message1);

producer.send(data);

data = new KeyedMessage<String, String>(topic, key2, message2);

producer.send(data);
```

Producer Side API: Example

Step 5: Close the connection

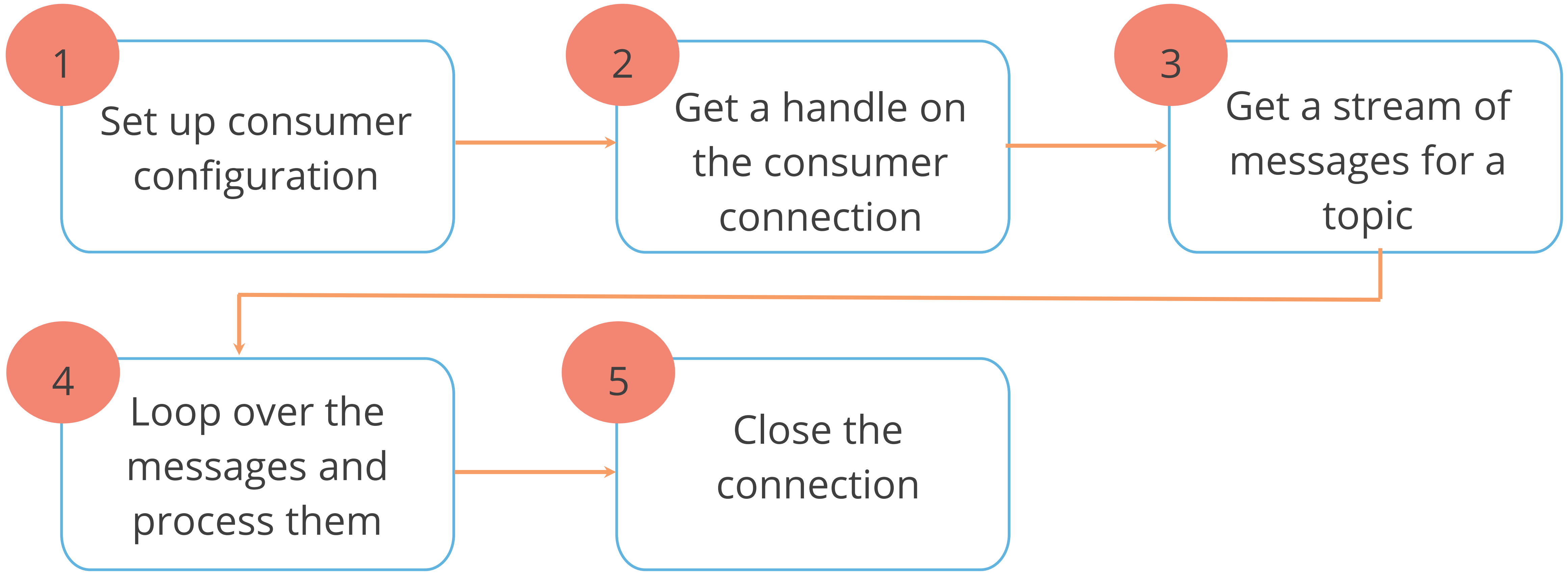

training@localhost:~

```
producer.close();
```

Consumer Side API

The consumer side APIs provide an interface to connect to the cluster and get messages from a topic.

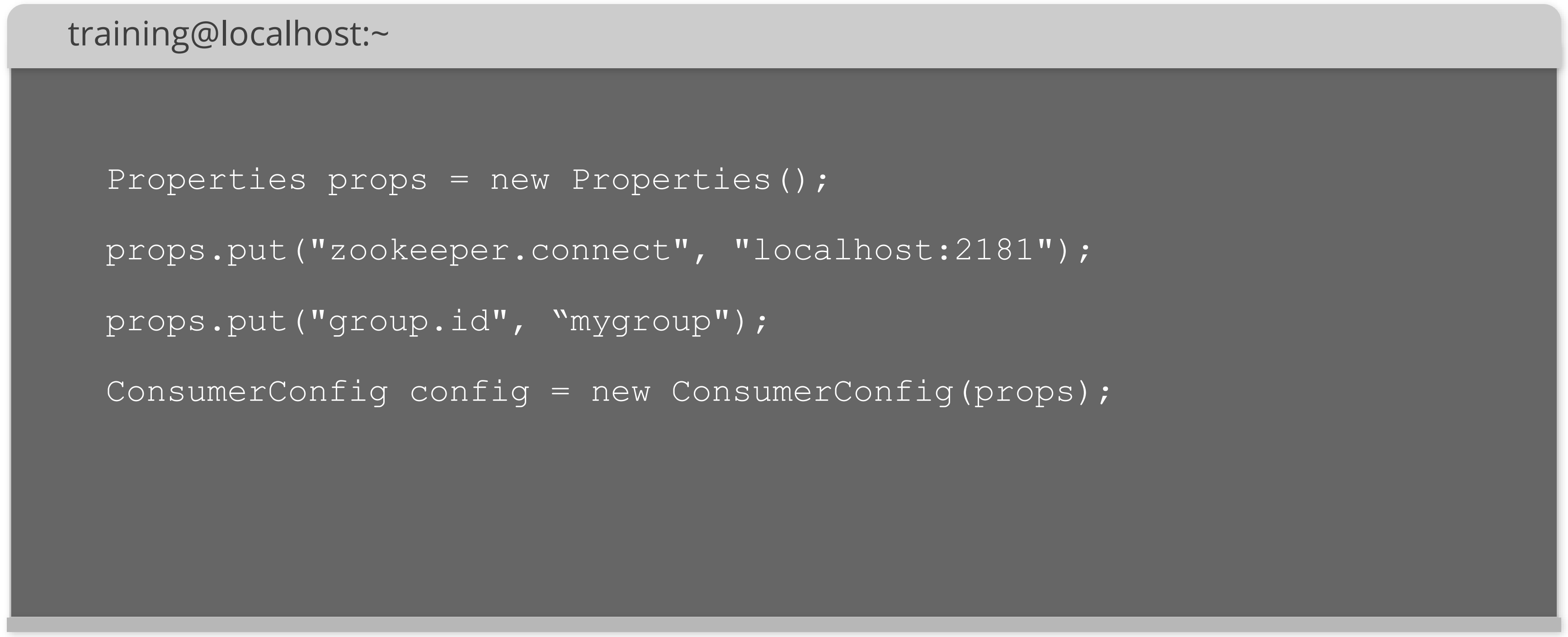
The steps involved in programming are:


1
Set up consumer configuration
2
Get a handle on the consumer connection
3
Get a stream of messages for a topic
4
Loop over the messages and process them
5
Close the connection

Consumer Side API: Example

The consumer connects with the Kafka broker hosts (worker nodes) in a loop and reads records.

Step 1: Set up consumer configuration

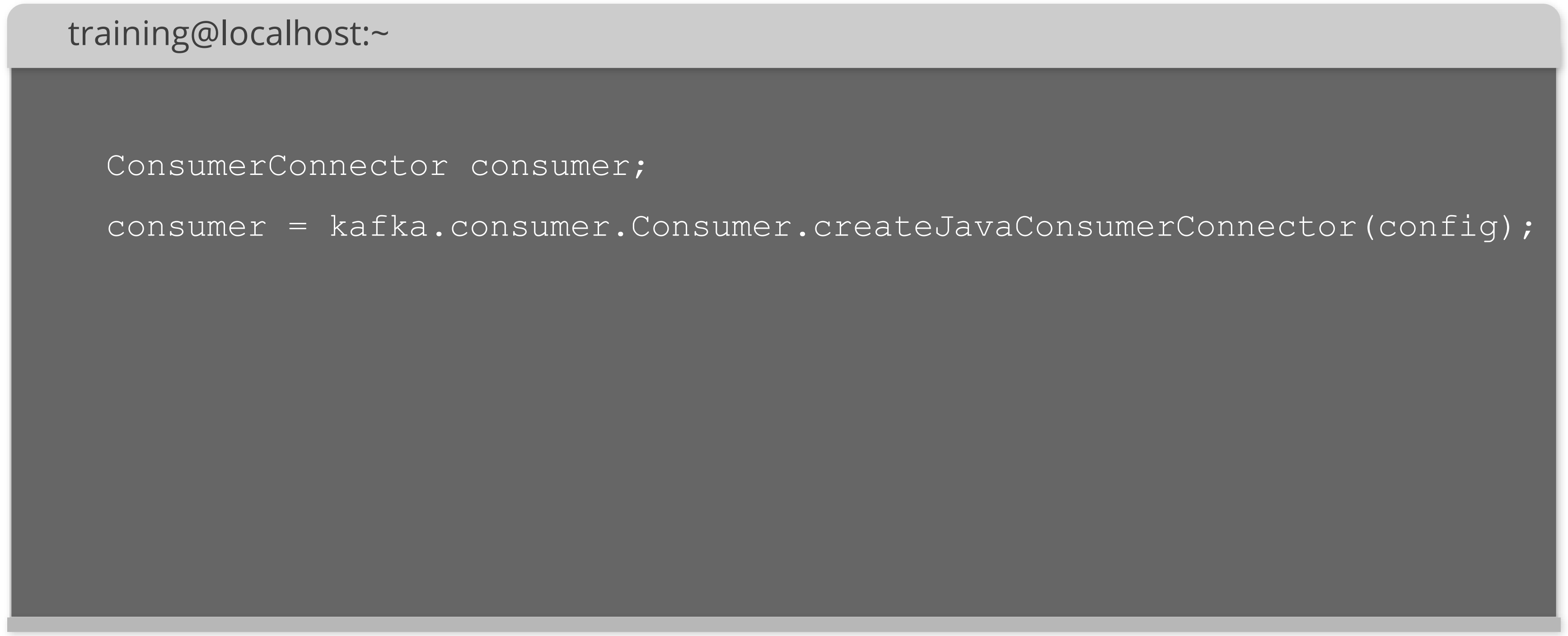


training@localhost:~

```
Properties props = new Properties();
props.put("zookeeper.connect", "localhost:2181");
props.put("group.id", “mygroup");
ConsumerConfig config = new ConsumerConfig(props);
```

Consumer Side API: Example

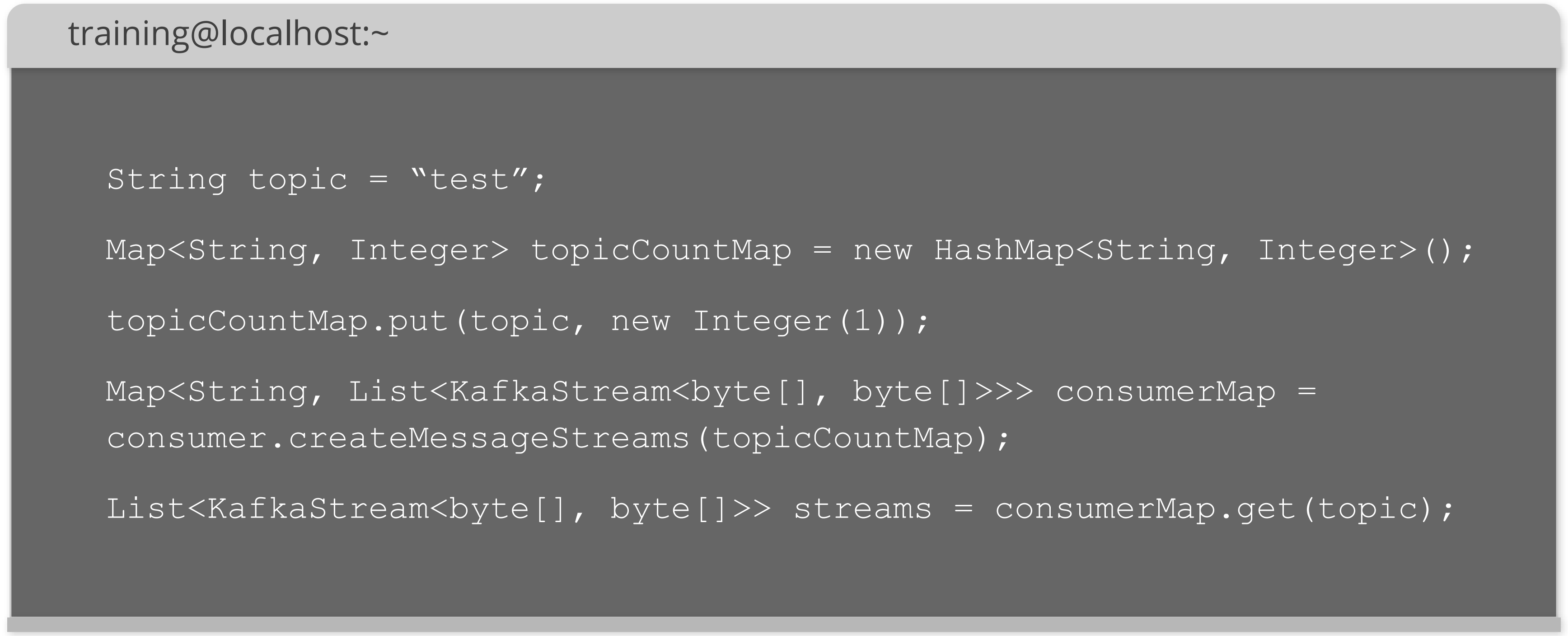
Step 2: Get a handle on the consumer connection


training@localhost:~

```
ConsumerConnector consumer;
consumer = kafka.consumer.Consumer.createJavaConsumerConnector(config);
```

Consumer Side API: Example

Step 3: Get a stream of messages for the topic

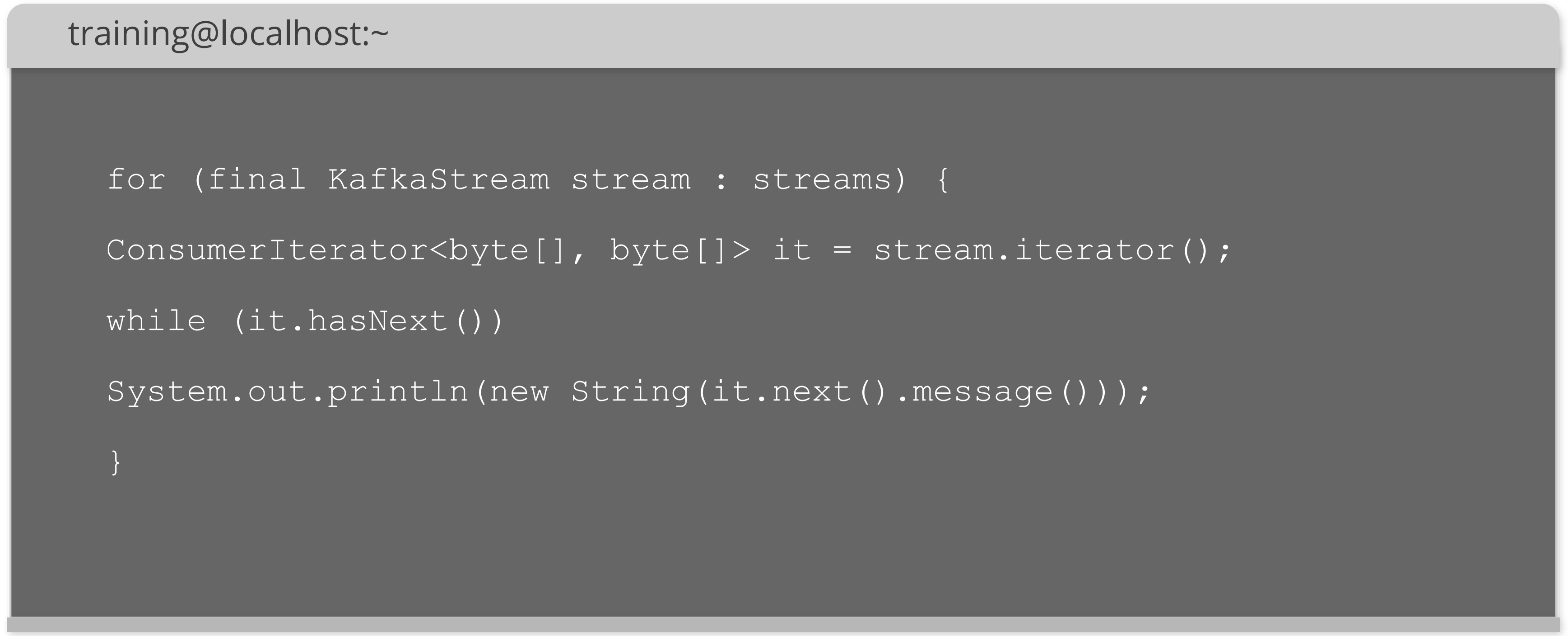


training@localhost:~

```
String topic = “test”;

Map<String, Integer> topicCountMap = new HashMap<String, Integer>();

topicCountMap.put(topic, new Integer(1));

Map<String, List<KafkaStream<byte[], byte[]>>> consumerMap =
consumer.createMessageStreams(topicCountMap);

List<KafkaStream<byte[], byte[]>> streams = consumerMap.get(topic);
```

Consumer Side API: Example

Step 4: Loop over the messages and process them

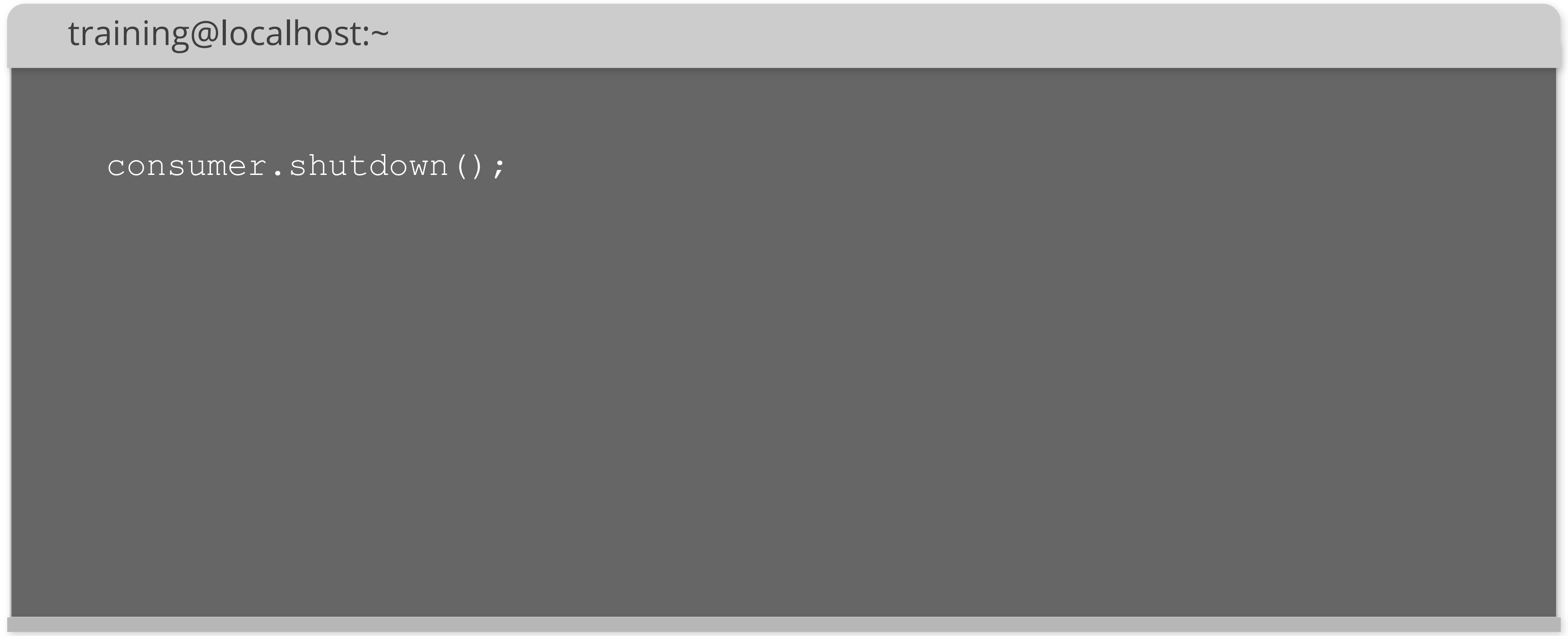


training@localhost:~

```
for (final KafkaStream stream : streams) {
ConsumerIterator<byte[], byte[]> it = stream.iterator();
while (it.hasNext())
System.out.println(new String(it.next().message()));
}
```

Consumer Side API: Example

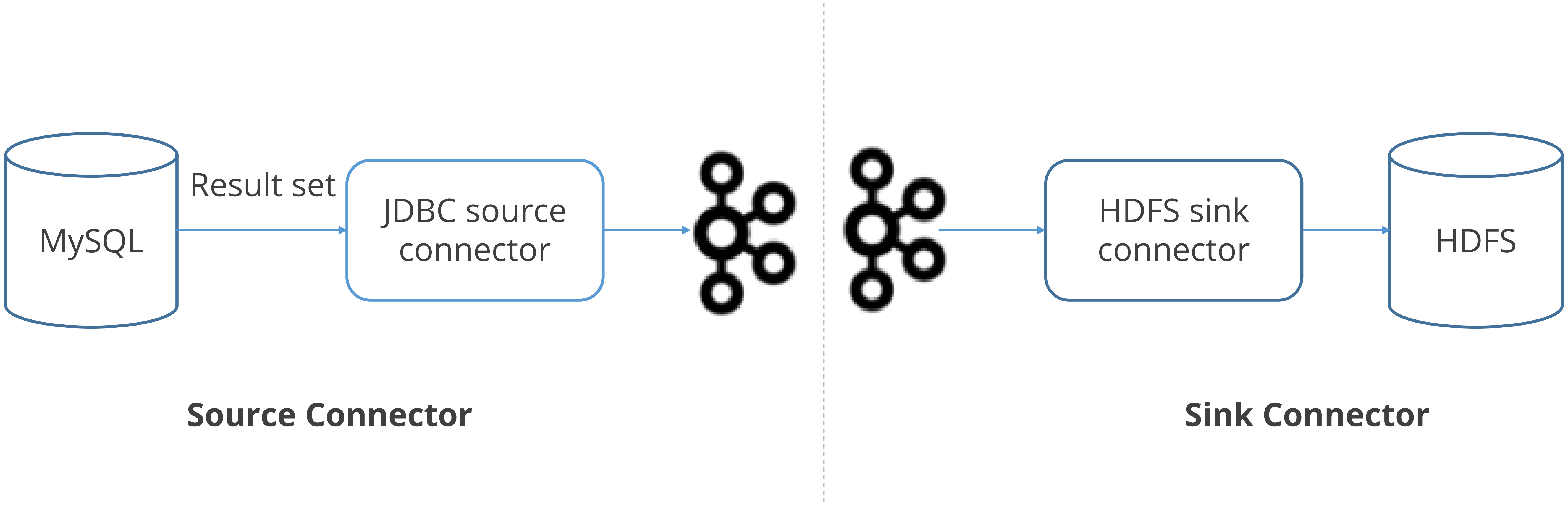
Step 5: Close the connection to the consumer


training@localhost:~

```
consumer.shutdown();
```

Kafka Connect

Kafka Connect is a framework for connecting Kafka with external systems such as databases, key-value stores, search indexes, and file systems using **Connectors**.



Confluent Connector

The confluent connector is an alternative to Kafka connect. It has some additional tools and clients, compared to plain Kafka, as well as some additional pre-built connectors.

Kafka Connect JDBC

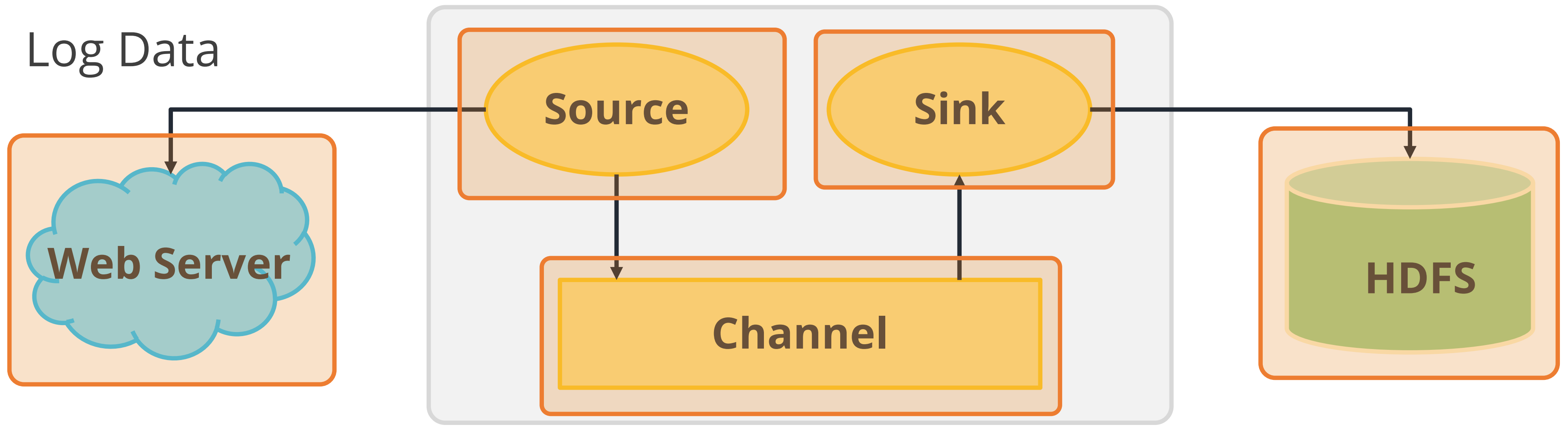
Kafka Connect S3

SAP Hana Connector

Apache Flume

What Is Apache Flume?

Apache Flume is a distributed and dependable service for gathering, aggregating, and transporting massive volumes of streaming data into the Hadoop Distributed File System effectively (HDFS).



Why Flume?

The following is a business scenario in which Flume is beneficial:

Current scenario **Problem** **Solution**

Thousands of services operating on separate servers in a cluster create a tremendous amount of log data, which should be evaluated collectively.

Why Flume?

The following is a business scenario in which Flume is beneficial:

Current scenario

Problem

Solution

The current issue involves determining how to send the logs to a setup that has Hadoop. The channel or method used for the sending process must be reliable, scalable, extensive, and manageable.

Why Flume?

The following is a business scenario in which Flume is beneficial:

Current scenario

Problem

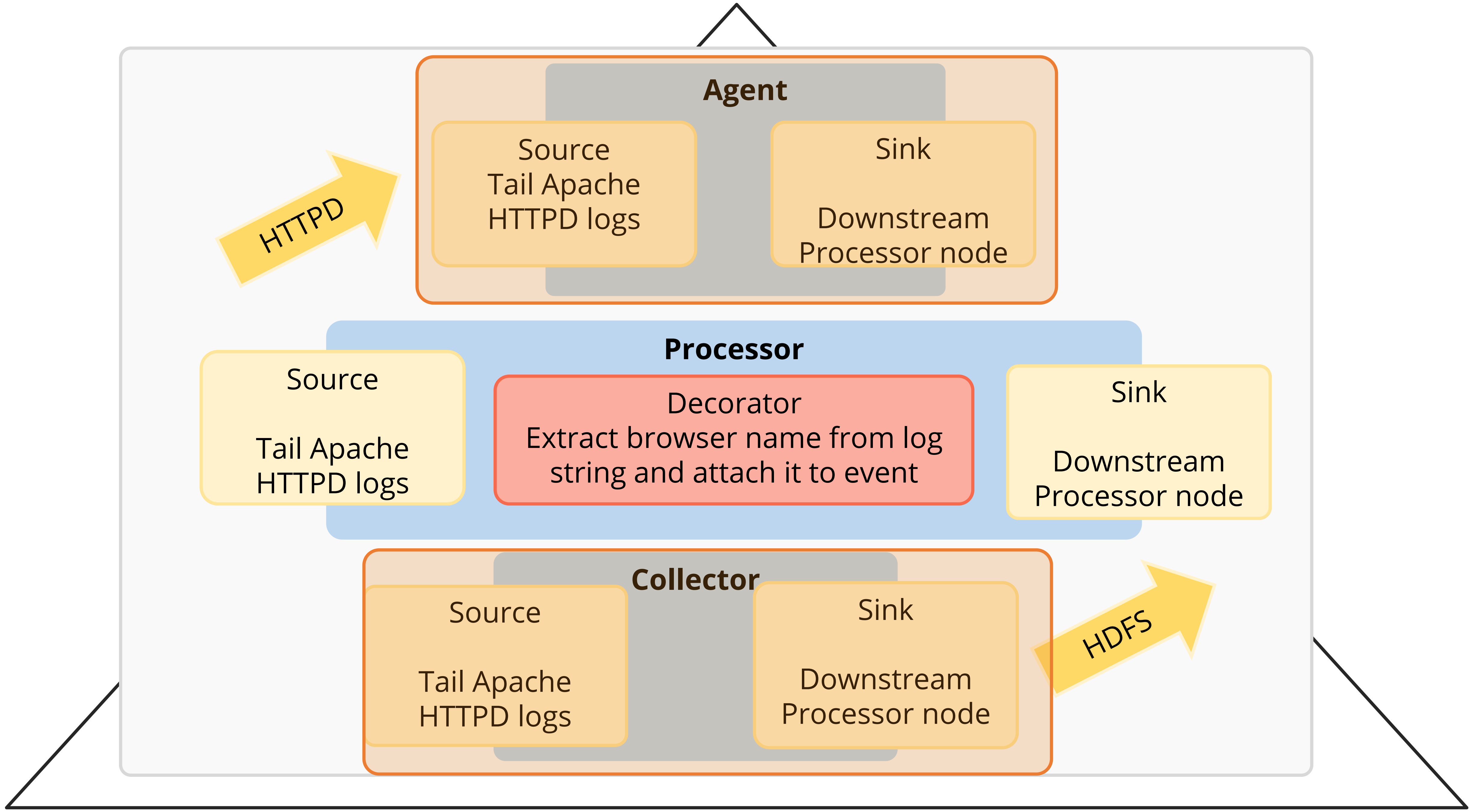
Solution

To solve this problem, a log aggregation tool called Flume can be used. Apache Sqoop and Flume are the tools that gather data from different sources and load them into HDFS.

Apache Flume Model

Flume Model

The Flume Model comprises the following three entities:

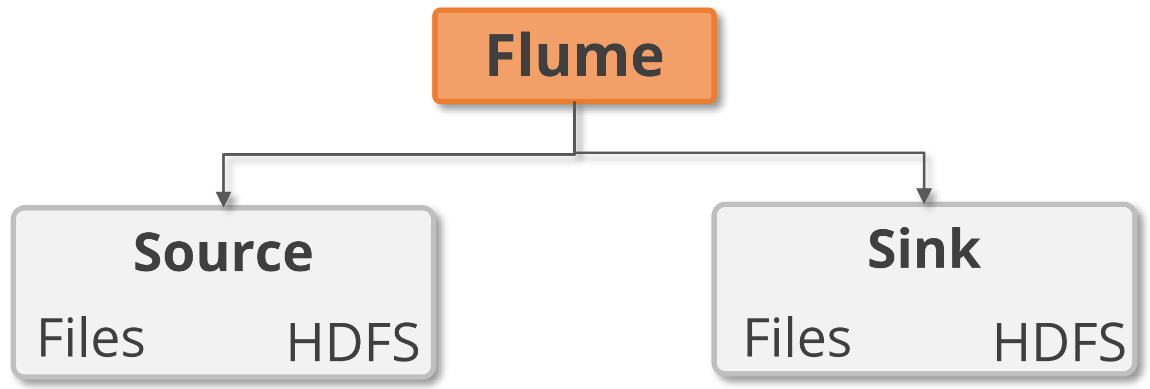


Flume Goals

Flume aims to achieve the following goals:

- Ensure reliability by possessing tunable failure recovery modes
- Attain extensibility by using plug-in architecture for extending modules
- Achieve a scalable data path to form a topology of agents
- Create manageability by centralizing a data flow management interface

Extensibility in Flume

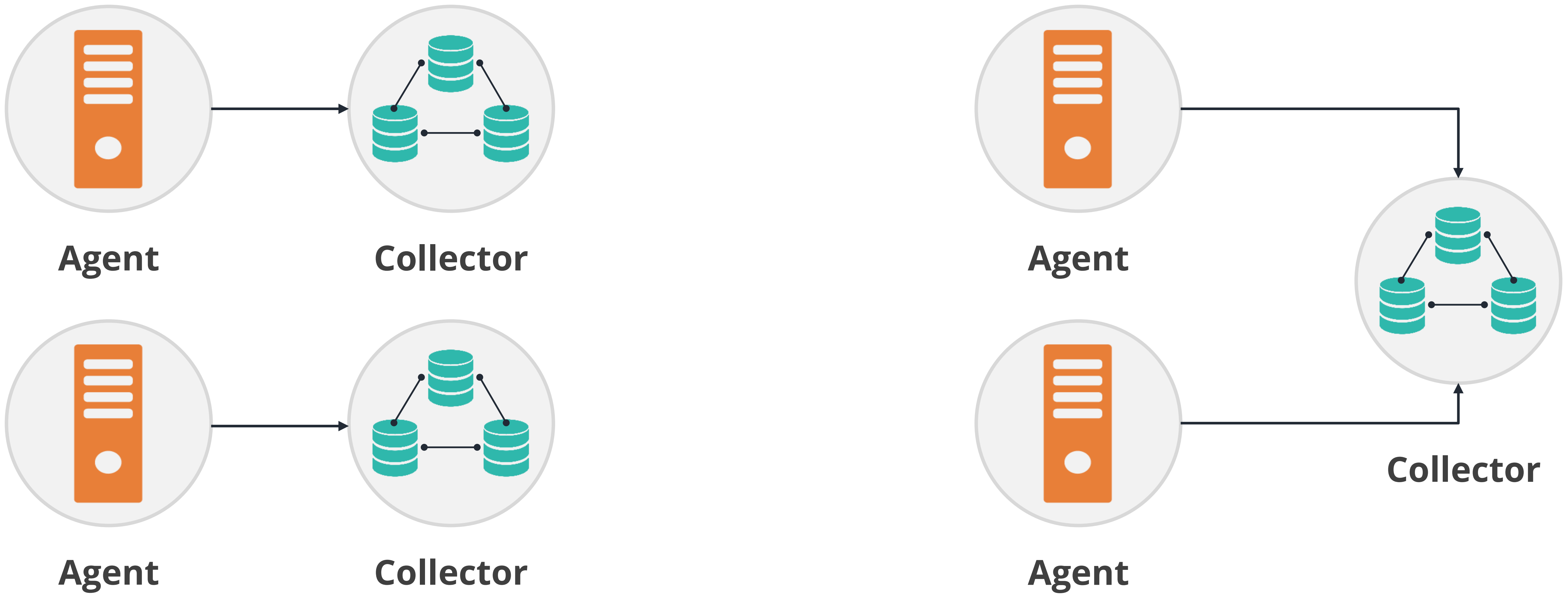


Extensibility:

- The ability to add new functionality to a system is called extensibility.
- Flume can be extended by adding sources and sinks to existing storage layers or data platforms.
- General sources include data from files, Syslog, and standard output from any Linux process.
- General sinks include files on the local filesystem or HDFS.
- Developers can write their own sources or sinks.

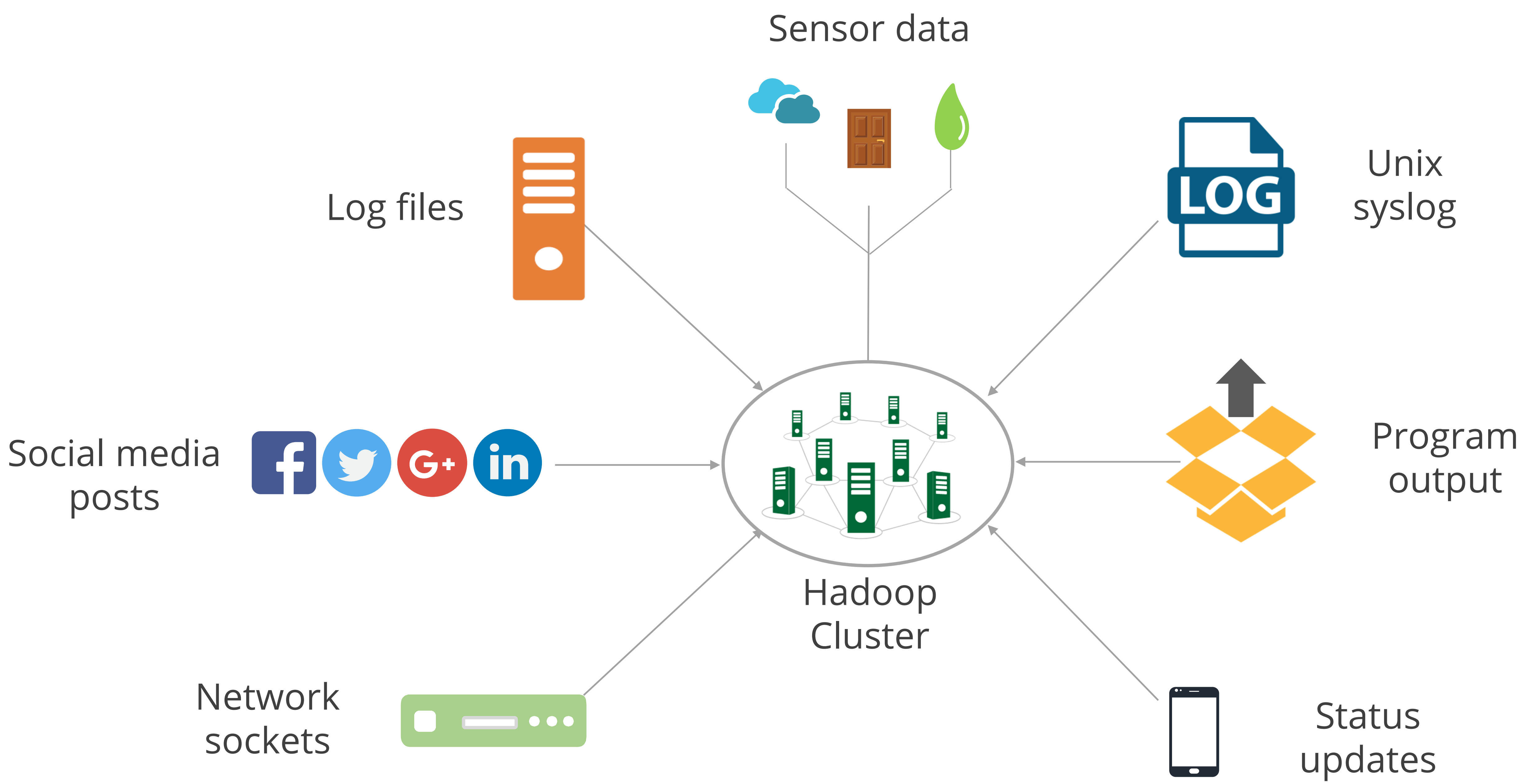
Scalability in Flume

Flume has a horizontally scalable data path which helps in achieving load balance in case of higher load in the production environment.



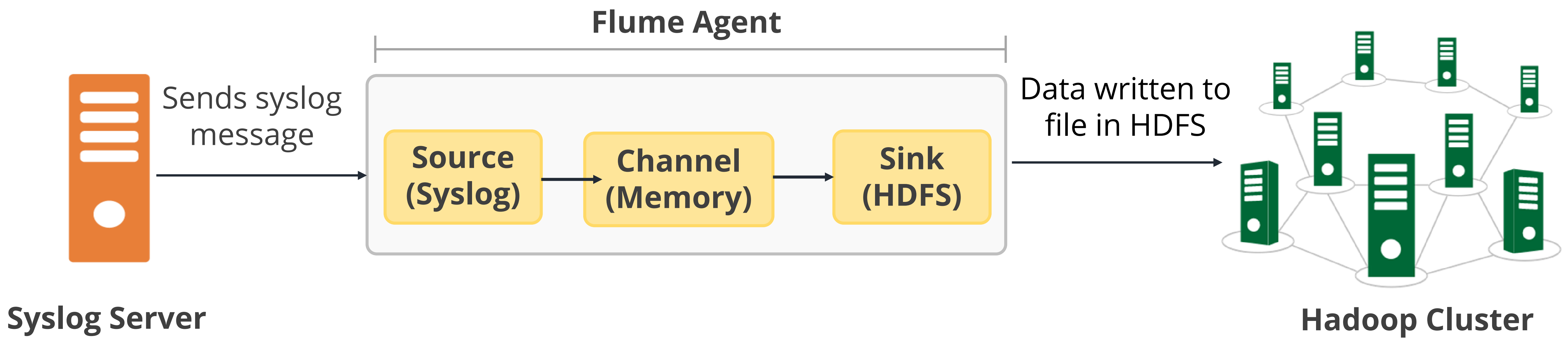
Common Flume Data Sources

The following diagram depicts the common flume data sources:



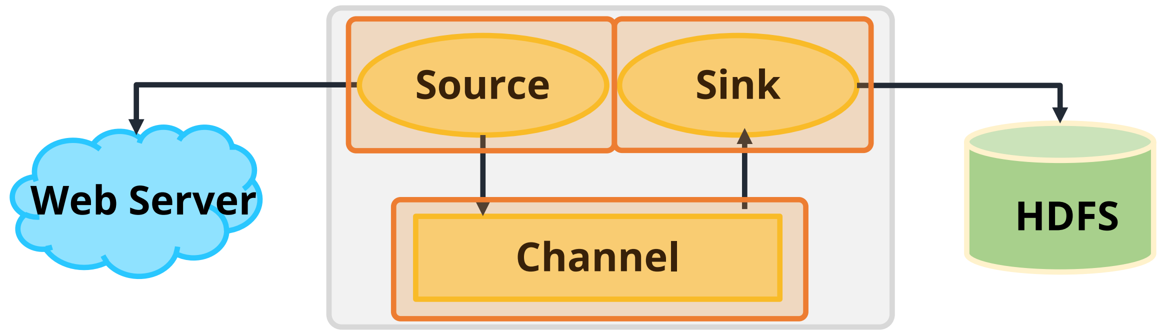
Flume Data Flow

The diagram illustrates how Syslog data is captured to HDFS.



Components in Flume’s Architecture

Components in Flume's Architecture



- **Source:** Events are received from the external actor who creates them.
- **Sink:** It sends an event to its destination and saves the data in centralized storage, such as HDFS and HBASE.
- **Channel:** Events are buffered from the source until they are drained by the sink.
- **Agent:** A Java process configures and hosts source, channel, and sink

Flume Source

The Apache Flume source is a Flume agent component that takes data from external sources and routes it to one or more channels.

Netcat: Listens on a given port and turns each line of text into an event

Kafka: Receives events as messages from a Kafka topic

Syslog: Captures messages from UNIX Syslog daemon over the network

Spooldir: Used for ingesting data by placing files to be ingested into a "spooling“ directory on the disk

Flume Sink

The following are the types of flume sink:

Null
Discards all events received
(Flume equivalent of **/dev/null**)

HBaseSink
Stores event in HBase

HDFS
Writes event to a file in the specified directory in HDFS

Flume Channels

The following are the types of flume channels:

Memory

- Stores events in the machine's RAM
- Is extremely fast, but not reliable as memory is volatile

File

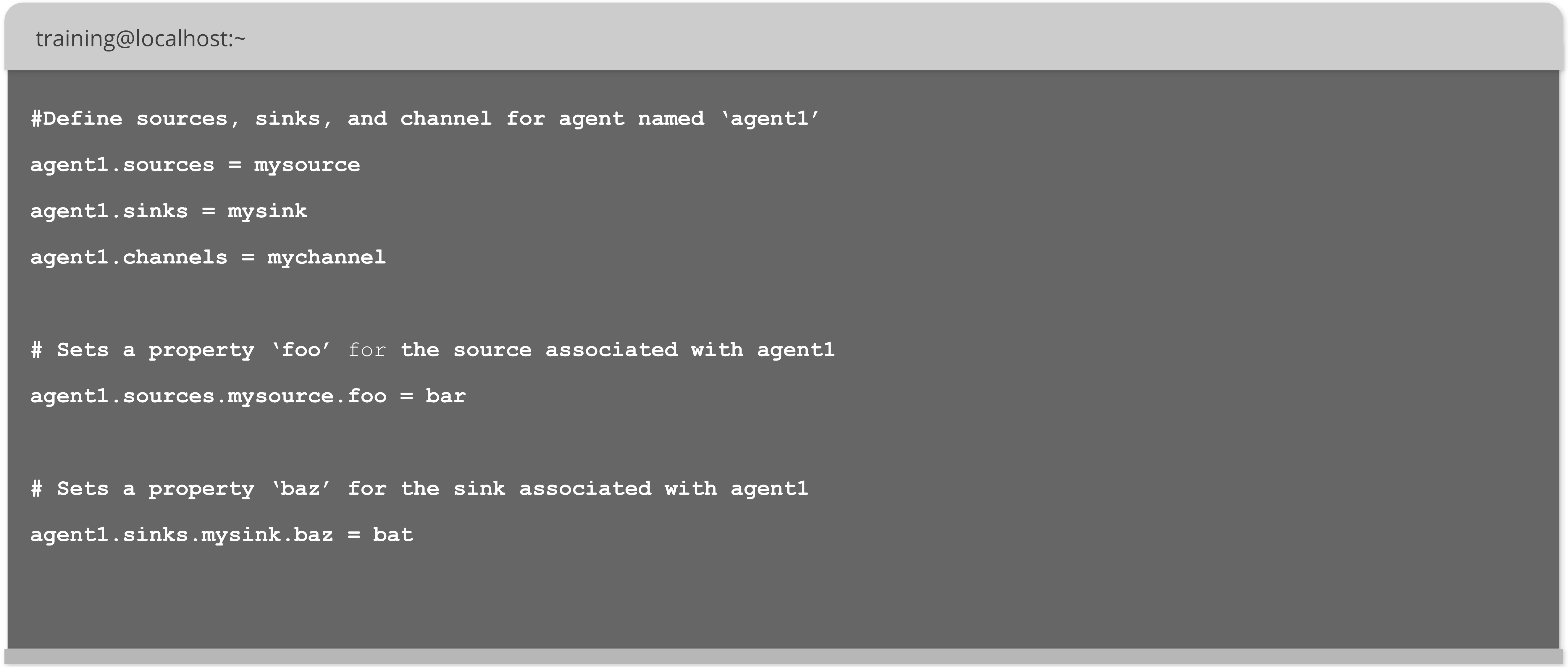
- Stores events on the machine's local disk
- Is slower than RAM, but more reliable as data is written to disk

JDBC

- Stores events in a database table using JDBC
- Is slower than file channel

Flume Agent Configuration File

All flume agents can be configured in a single Java file.

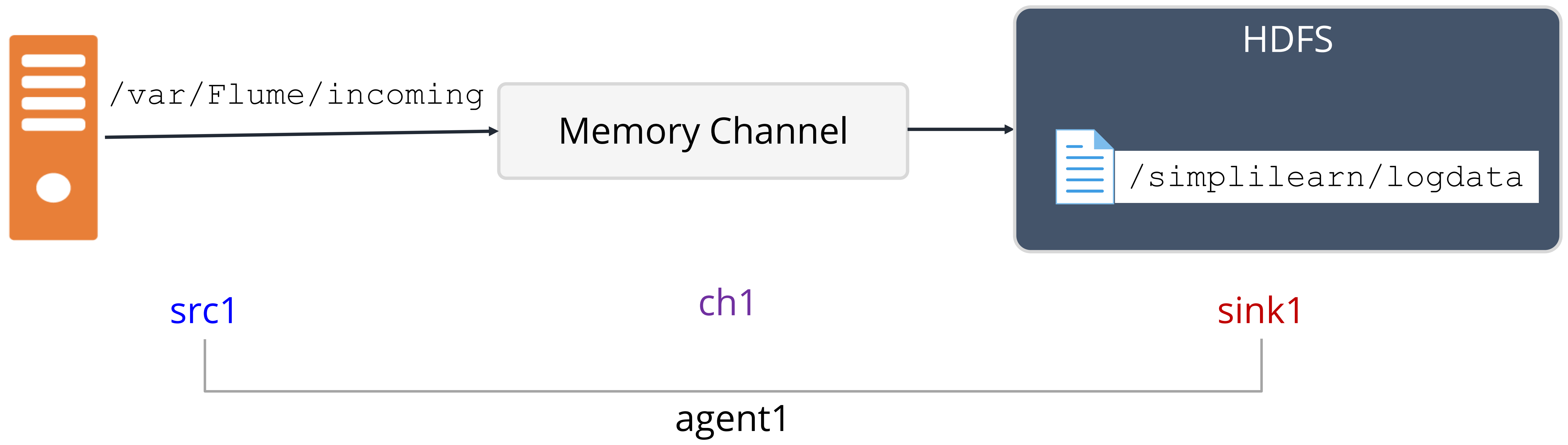


training@localhost:~

```
#Define sources, sinks, and channel for agent named ‘agent1’
agent1.sources = mysource
agent1.sinks = mysink
agent1.channels = mychannel

# Sets a property ‘foo’ for the source associated with agent1
agent1.sources.mysource.foo = bar

# Sets a property ‘baz’ for the sink associated with agent1
agent1.sinks.mysink.baz = bat
```

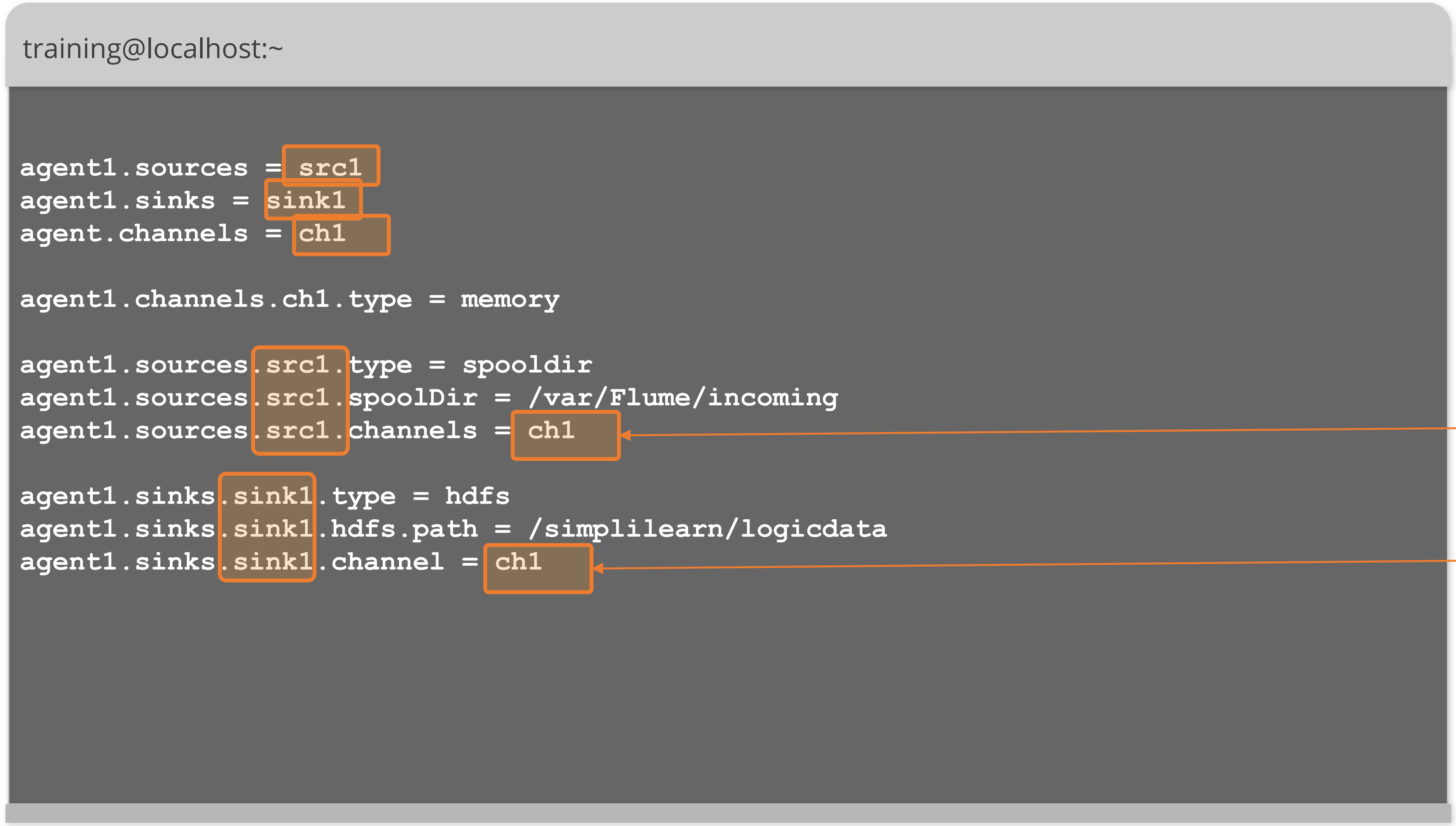
Configuring Flume Components: Example

The following diagram shows how to configure a Flume Agent to collect data from remote spool directories and save it to HDFS through a memory channel.



Configuring Flume Configuration: Example

The following example shows how to configure Flume Configuration:


training@localhost:~

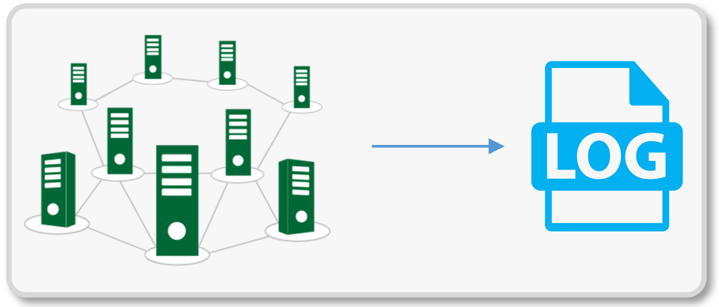
```
agent1.sources = src1
agent1.sinks = sink1
agent.channels = ch1

agent1.channels.ch1.type = memory

agent1.sources.src1.type = spooldir
agent1.sources.src1.spoolDir = /var/Flume/incoming
agent1.sources.src1.channels = ch1

agent1.sinks.sink1.type = hdfs
agent1.sinks.sink1.hdfs.path = /simplilearn/logicdata
agent1.sinks.sink1.channel = ch1
```

Connect **source** and channel

Connect **source** and channel

Flume: Sample Use Cases

Flume can be used for a variety of use cases:



To collect logs from nodes in Hadoop cluster

To collect logs from services such as http and mail

For process monitoring

To collect impressions from custom applications for an advertisement network

Key Takeaways

- Big data ingestion is transporting data, particularly unstructured data, from its source into a system like Hadoop where it can be stored and analyzed.
- The architecture of Kafka, including the producer, consumer, and cluster, is summarized.
- Apache Flume is a distributed and dependable service for gathering, aggregating, and transporting massive volumes of streaming data into the HDFS.
- The Apache Flume source is a Flume agent component that takes data from external sources and routes it to one or more channels.

Knowledge Check

Knowledge Check

1

Which of the following is a Flume source?

A. Null

B. HDFS

C. Spooldir

D. Sink

Knowledge Check 1

Which of the following is a Flume source?

A. Null

B. HDFS

C. Spooldir

D. Sink

The correct answer is **C**

Spooldir is a Flume source.

Knowledge Check

2

______ is fault-tolerant, linearly scalable, and stream-oriented.

A. Sqoop

B. Flume

C. Kafka

D. All of the above

Knowledge Check
2

______ is fault-tolerant, linearly scalable, and stream-oriented.

A. Sqoop

B. Flume

C. Kafka

D. All of the above

The correct answer is **B**

Flume is a flexible data ingestion tool that is fault-tolerant, linearly scalable, and stream-oriented.

Knowledge Check

3

The parallelism of a Kafka topic can be set using the parameter:

A. Replication factor

B. Partitions

C. Threads

D. Concurrent

Knowledge Check

3

The parallelism of a Kafka topic can be set using the parameter:

A. Replication factor

B. Partitions

C. Threads

D. Concurrent

The correct answer is **B**

The number of partitions determines the parallelism of a topic in Kafka and is set by using the –partitions option in the create-topic.sh command.

Knowledge Check

4

How does the Kafka console consumer connect to the Kafka cluster?

A. By using the list of Kafka servers provided on the command line

B. By using the list of ZooKeeper servers provided on the command line

C. By reading the configuration file

D. By connecting to the local host

Knowledge Check

4

How does the Kafka console consumer connect to the Kafka cluster?

A. By using the list of Kafka servers provided on the command line

B. By using the list of ZooKeeper servers provided on the command line

C. By reading the configuration file

D. By connecting to the local host

The correct answer is **B**

The Kafka console consumer expects the list of ZooKeeper servers on the command line to connect to the Kafka cluster.

Thank You