Introduction to Apache Kafka
Apache Kafka

Chapter 1
Course Chapters

- Apache Kafka
- Integrating Flume and Kafka
Apache Kafka

In this chapter, you will learn

- What Kafka is and what advantages it offers
- About the high-level architecture of Kafka
- What several use cases for Kafka are
- How to create topics, publish messages, and read messages from the command line and in Java code
Chapter Topics

Apache Kafka

- Overview
- Use Cases
- Messages, Topics, and Partitions
- Producers and Consumers
- Message Ordering Guarantees
- Using the Java API
- Essential Points
- Hands-On Exercise: Using Kafka from the Command Line
What is Apache Kafka?

- **Apache Kafka is a distributed commit log service**
  - Widely used for data ingest
  - Offers scalability, performance, reliability, and flexibility
  - Conceptually similar to a publish-subscribe messaging system

- **Originally created at LinkedIn, but now an open source Apache project**
  - Donated to the Apache Software Foundation in 2012
  - Graduated from the Apache Incubator in 2013
  - Included as part of Cloudera Labs in 2014
  - Supported by Cloudera for production use with CDH in 2015
Characteristics of Kafka

- **Scalable**
  - Kafka is a distributed system that supports multiple nodes

- **Fault-tolerant**
  - Data is persisted to disk and replicated throughout the cluster

- **High throughput**
  - Each broker can process hundreds of thousands of messages per second

- **Low latency**
  - Data is delivered in a fraction of a second

- **Flexible**
  - Decouples the production of data from its consumption

*Using modest hardware, with messages of a typical size*
High-Level Architecture: Terminology

- **Messages** represent arbitrary user-defined content
  - For example, application events or sensor readings

- **A node running the Kafka service is called a broker**
  - A production cluster typically has many Kafka brokers
  - Kafka also depends on the ZooKeeper service for coordination

- **Producers** push messages to a broker
  - The producer assigns a *topic*, or category, to each message

- **Consumers** pull messages from a Kafka broker
  - They read only messages in relevant topics
High-Level Architecture: Example

Producer #1

Producer #2

Producer #3

Broker

Broker

Broker

Broker

Broker

Broker

Consumer #1

Consumer #2
Chapter Topics

Apache Kafka

- Overview
- Use Cases
  - Messages, Topics, and Partitions
  - Producers and Consumers
  - Message Ordering Guarantees
  - Using the Java API
- Essential Points
- Hands-On Exercise: Using Kafka from the Command Line
Why Kafka?

- Kafka is used for a variety of use cases, such as
  - Log aggregation
  - Messaging
  - Web site activity tracking
  - Operational metrics
  - Stream processing
  - Event sourcing

- A subset of these could also be done with Flume
  - For example, aggregating Web server log data into HDFS

- Kafka often becomes a better choice as use case complexity grows
Common Kafka Use Cases (1)

- **Distributed message bus / central data pipeline**
  - Enables highly scalable EAI, SOA, CEP and microservice architectures
  - Decouples services with a standardized message abstraction
  - Supports multiple message client languages with high throughput

- **Log aggregation**
  - Kafka can collect logs from multiple services
  - Logs can be made available to multiple consumers, such as Hadoop and Apache Solr

EAI: Enterprise Application Integration
SOA: Service-Oriented Architecture
CEP: Complex Event Processing
Common Kafka Use Cases (2)

- **Web site activity tracking**
  - Web application sends events such as page views and searches to Kafka
  - Events become available for real-time processing, dashboards, and offline analytics in Hadoop

- **Alerting and reporting on operational metrics**
  - Kafka producers and consumers occasionally publish their message counts to a special Kafka topic
  - A service compares counts and sends an alert upon detecting data loss

- **Stream processing**
  - A framework such as Spark Streaming reads data from a topic, processes it, and writes processed data to a new topic where it becomes available for users and applications
  - Kafka’s strong durability helps to facilitate this use case
Chapter Topics

Apache Kafka

- Overview
- Use Cases
- Messages, Topics, and Partitions
- Producers and Consumers
- Message Ordering Guarantees
- Using the Java API
- Essential Points
- Hands-On Exercise: Using Kafka from the Command Line
Messages

- Messages in Kafka are variable-size byte arrays
  - Allows for serialization of data in any format your application requires
  - Common formats include strings, JSON, and Avro

- There is no explicit limit on message size
  - Optimal performance usually occurs with messages of a few KB in size
  - We recommend that you do not exceed 1MB per message

- Kafka retains all messages for a defined time period
  - This period can be set on global or per-topic basis
  - Messages will be retained regardless of whether they were read
  - They are discarded automatically after the retention period is exceeded
Topics

- There is no explicit limit on the number of topics
  - Kafka works better with a few large topics than many small ones

- A topic can be created explicitly or simply by publishing to the topic
  - Controlled by the `auto.create.topics.enable` property
  - We recommend that topics be created explicitly
Topic Partitioning

- Each topic is divided into some number of partitions †
  - Partitioning improves scalability and throughput

- A topic partition is an ordered and immutable sequence of messages
  - New messages are appended to the partition as they are received
  - Each message is assigned a unique sequential ID known as an offset

† Note that this is unrelated to partitioning in HDFS, MapReduce, or Spark
Replication

- Each partition can be replicated across a configurable number of brokers‡
  - Doing so is recommended, as it provides fault tolerance

- Each broker acts as a leader for some partitions and a follower for others
  - Followers passively replicate the leader
  - If the leader fails, a follower will automatically become the new leader

‡Note that this is unrelated to HDFS replication
Starting the Kafka Broker

- In production, you will likely start Kafka via Cloudera Manager
  - In this class, we must start it manually on the VM

- Since Kafka depends on ZooKeeper, we must start that service first
  
  ```
  $ sudo service zookeeper-server start
  ```

- We can then start the Kafka service
  
  ```
  $ sudo service kafka-server start
  ```
Creating Topics from the Command Line

- **Kafka includes a convenient set of command line tools**
  - These are helpful for exploring and experimentation

- **The kafka-topics command offers a simple way to create Kafka topics**
  - Provide the topic name of your choice, such as `device_status`
  - You must also specify the ZooKeeper connection string for your cluster

```bash
$ kafka-topics --create \
  --zookeeper localhost:2181 \
  --replication-factor 1 \
  --partitions 1 \
  --topic device_status
```
Displaying Topics from the Command Line

- Use the `--list` parameter to list all topics

```bash
$ kafka-topics --list --zookeeper localhost:2181
```
Chapter Topics

Apache Kafka

- Overview
- Use Cases
- Messages, Topics, and Partitions
- **Producers and Consumers**
- Message Ordering Guarantees
- Using the Java API
- Essential Points
- Hands-On Exercise: Using Kafka from the Command Line
Producer Recap

- Producers publish messages to Kafka topics
  - They communicate with a broker, not a consumer
Selecting the Partition

- A producer is responsible for selecting partitions for messages it publishes
  - This is primarily done to balance the load across all partitions
  - The producer writes messages to a partition in order
  - A pluggable Partitioner class selects the partition for each message

![Partition Diagram]

Copyright © 2010–2017 Cloudera. All rights reserved. Not to be reproduced or shared without prior written consent from Cloudera.
Aside: Message Batches Increase Throughput and Latency

- **Producers can collect multiple messages to write to a partition**
  - This reduces the number of requests made to brokers
  - Such requests sent to brokers contain one batch per partition

- **Batching is controlled through properties set for the producer**
  - The default is to send messages immediately
  - Batch size is configurable, as is the max time to wait before sending

![Diagram showing message batches and time]
Messages are Replicated

- The producer is configured with a list of one or more brokers
  - It asks the first available broker for the leader of the desired partition

- The producer then sends the message to the leader
  - The leader writes the message to its local log
  - Each follower then writes the message to its own log
  - After acknowledgements from followers, the message is committed
Creating a Producer from the Command Line (1)

- You can create a producer using the `kafka-console-producer` tool.

- Specify one or more brokers in the `--broker-list` option:
  - Each broker consists of a hostname, a colon, and a port number.
  - If specifying multiple brokers, separate them with commas.
  - In our case there is one broker: `localhost:9092`.

- You must also provide the name of the topic:
  - We will publish messages to the topic named `device_status`.

```sh
$ kafka-console-producer \
   --broker-list localhost:9092 \
   --topic device_status
```
Creating a Producer from the Command Line (2)

- You may see a few log messages in the terminal after the producer starts.
- It will then accept input in the terminal window:
  - Each line you type will be a message sent to the topic.
- Until you have configured a consumer for this topic, you'll see no other output from Kafka.
Consumer Recap

- A consumer reads messages that were published to Kafka topics
  - They communicate with a broker, not a producer

- Consumer actions do not affect other consumers
  - For example, issuing the Kafka command line tool to “tail” the contents of a topic does not change what is consumed by other consumers

- They can come and go without impact on the cluster or other consumers
Creating a Consumer from the Command Line

- You can create a consumer with the `kafka-console-consumer` tool.
- This requires the ZooKeeper connection string for your cluster.
  - Unlike creating a producer, which instead required a list of brokers.
- The command also requires a topic name.
  - In our case, we will use `device_status`.
- You can use `--from-beginning` to read all available messages.
  - Otherwise, it would read only new messages.

```
$ kafka-console-consumer \
  --zookeeper localhost:2181 \
  --topic device_status \
  --from-beginning
```
Writing File Contents to Topics via the Command Line

- Using UNIX pipes or redirection, you can read input from files
  - The data can then be sent to a topic using the command line producer

- This example shows how to read input from a file named alerts.txt
  - Each line in this file becomes a separate message in our topic

```bash
$ cat alerts.txt | kafka-console-producer \
  --broker-list localhost:9092 \
  --topic device_status
```

- This technique can be an easy way to integrate with existing programs
How does Kafka differ from traditional message models?

- Messaging has two traditional models
  - Queuing
  - Publish-subscribe

- With queuing, a pool of consumers may read from a server and each message goes to one of them

- In publish-subscribe, the message is broadcast to all consumers

- A Kafka consumer group is a consumer abstraction that generalizes both of these models
Kafka Consumer Group Operation

- Each message published to a topic is delivered to one consumer instance within each subscribing consumer group
- Consumer instances can be in separate processes or on separate machines
- The diagram below depicts a Kafka cluster with two broker (servers)
  - The brokers are hosting four partitions, P0-P3
  - Consumer group A has two consumer instances and group B has four
Kafka Consumer Group Configurations

- Kafka functions like a traditional queue when
  - All consumer instances belong to the same consumer group
  - In this case, a given message is received by one consumer

- Kafka functions like traditional publish-subscribe when
  - Each consumer instance belongs to a different consumer group
  - In this case, all messages are broadcast to all consumers
Using “Logical Subscribers”

- In between the two extremes of queuing or publish-subscribe lies a balanced solution
  - A topic can have one consumer group for each “logical subscriber”

- In this approach, each consumer group is composed of many consumer instances
  - This provides scalability and fault tolerance
  - Amounts to publish-subscribe semantics where the subscriber is a cluster of consumers instead of a single process
Chapter Topics

Apache Kafka

- Overview
- Use Cases
- Messages, Topics, and Partitions
- Producers and Consumers
- Message Ordering Guarantees
- Using the Java API
- Essential Points
- Hands-On Exercise: Using Kafka from the Command Line
Traditional Message Ordering

- A traditional queue retains messages in order on the server
  - The server hands out messages to consumers in the order they are stored

- In some message systems, messages delivered to consumers asynchronously may arrive out of order at different consumers
  - Message order is effectively lost in the presence of parallel consumption

- The workaround is to allow only one process to consume from a queue
  - This is the “exclusive consumer” approach
  - There is no parallelism
Kafka Ordering

- Partitions within Kafka topics make it possible to provide a consumer group with
  - Message ordering guarantees
  - Load balancing

- Partitions are assigned to consumers in a consumer group
  - Each partition is consumed by exactly one consumer in the group
  - The consumer of a partition is the only reader of that partition and consumes the data in order

- The number of consumers cannot exceed the number of partitions
Kafka Ordering Tip

- Kafka only provides a total order over messages within a partition, not between different partitions in a topic
- Per-partition ordering combined with the ability to partition data by key is sufficient for most applications
- Some applications require total ordering for a given topic
  - Accomplish this by creating just one partition for the topic
  - Note that this means only one consumer process is allowed
Kafka Guarantees

- Messages sent by a producer to a particular topic partition will be appended in the order they are sent
  - For example, if message M1 is sent by the same producer as message M2, and M1 is sent first, then
    - M1 will have a lower offset than M2
    - M1 will appear earlier in the log than M2

- A consumer sees messages in the order in which they are stored in the log

- For a topic with replication factor N, up to N-1 server failures can occur without losing any messages committed to the log
Chapter Topics

Apache Kafka

- Overview
- Use Cases
- Messages, Topics, and Partitions
- Producers and Consumers
- Message Ordering Guarantees
- **Using the Java API**
- Essential Points
- Hands-On Exercise: Using Kafka from the Command Line
Kafka Java API: Producer

- Kafka’s Java API allows you to easily create producers and consumers
  - Your code can send messages to a topic using a producer
  - Your code can also read messages sent to a topic using a consumer

- The next three slides show sample code for a simple producer that sends a message to a topic
package com.loudacre.example;

import java.util.Properties;
import java.util.concurrent.Future;

import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.Producer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.common.serialization.StringSerializer;

public class ProducerExample {

    public static void main(String[] args) {

        Note: file continues on next slide
Properties props = new Properties();

// This is a comma-delimited list of brokers to contact
props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
    "localhost:9092");

// This specifies that the write will only be committed
// after all brokers with replicas have acknowledged it
props.put(ProducerConfig.ACKS_CONFIG, "all");

// # of bytes to collect in message batch before sending
props.put(ProducerConfig.BATCH_SIZE_CONFIG, 16384);

// Specifies classes used for message serialization
props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
    StringSerializer.class.getName());
props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
    StringSerializer.class.getName());
Simple Producer (3): Message Creation and Publication

```java
// Create a Producer using our configuration properties
Producer<String, String> producer =
    new KafkaProducer<String, String>(props);

// Specify the topic and value for the message
String topic = "app_events";
String value = "CART_ADD,alice,0872584";

// Create and send the message
ProducerRecord<String, String> message =
    new ProducerRecord<String, String>(topic, value);

producer.send(message);

// Close the producer once we no longer need it
producer.close();
```
Kafka Java API: Consumer

- The next few slides provide sample code for a simple consumer
  - This consumer reads messages posted to the selected topic
package com.loudacre.example;

import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Properties;

import kafka.consumer.Consumer;
import kafka.consumer.ConsumerConfig;
import kafka.consumer.ConsumerIterator;
import kafka.consumer.KafkaStream;
import kafka.javaapi.consumer.ConsumerConnector;
import kafka.serializer.Decoder;
import kafka.serializer.StringDecoder;

public class ConsumerExample {
    public static void main(String[] args) { timezone
    
    Note: file continues on next slide
// Define required properties and configure the consumer
Properties props = new Properties();
props.put("zookeeper.connect", "localhost:2181");
props.put("group.id", "example");
ConsumerConfig cfg = new ConsumerConfig(props);
ConsumerConnector consumer =
    Consumer.createJavaConsumerConnector(cfg);

// Prepare to subscribe to app_events with one thread
String topic = "app_events";
Map<String, Integer> tpx=new HashMap<String, Integer>();
    tpx.put(topic, Integer.valueOf(1));

// Set up the message decoder and subscribe to the topic
Decoder<String> dec = new StringDecoder(null);
Map<String, List<KafkaStream<String, String>>> sm =
    consumer.createMessageStreams(tpx, dec, dec);

Note: file continues on next slide
// Get our topic's stream and iterate over its messages
for (KafkaStream<String, String> strm : sm.get(topic)) {
    ConsumerIterator<String, String> i = strm.iterator();

    // Process each incoming message
    while (i.hasNext()) {
        String message = i.next().message();
        System.out.println("Message was: " + message);
    }
}

Chapter Topics

Apache Kafka

- Overview
- Use Cases
- Messages, Topics, and Partitions
- Producers and Consumers
- Message Ordering Guarantees
- Using the Java API

- Essential Points

- Hands-On Exercise: Using Kafka from the Command Line
Essential Points

- Nodes running the Kafka service are called brokers
- Producers publish messages to categories called topics
- Messages in a topic are read by consumers
  - Multiple consumer instances can belong to a consumer group
  - Kafka retains messages for a defined (but configurable) amount of time
  - Consumers maintain an offset to track which messages they have been read
- Topics are divided into partitions for performance and scalability
  - These partitions are replicated for fault tolerance
Bibliography

The following offer more information on topics discussed in this chapter

- The Apache Kafka Web site
  - http://kafka.apache.org/

- Real-Time Fraud Detection Architecture
  - http://tiny.cloudera.com/kmc01a

- Kafka Reference Architecture
  - http://tiny.cloudera.com/kmc01b

- The Log: What Every Software Engineer Should Know...
  - http://tiny.cloudera.com/kmc01c
Chapter Topics

Apache Kafka

- Overview
- Use Cases
- Messages, Topics, and Partitions
- Producers and Consumers
- Message Ordering Guarantees
- Using the Java API
- Essential Points
- Hands-On Exercise: Using Kafka from the Command Line
Hands-On Exercise: Using Kafka from the Command Line

- In this exercise, you will use Kafka’s command line utilities to create a new topic, publish messages to the topic with a producer, and read messages from the topic with a consumer
  - Please refer to the Hands-On Exercise Manual for instructions
Integrating Flume and Kafka

Chapter 2
Course Chapters

- Apache Kafka
- Integrating Flume and Kafka
Integrating Flume and Kafka

In this chapter, you will learn

▪ What to consider when choosing between Flume and Kafka for a use case
▪ How Flume and Kafka can work together
▪ How to configure a Flume source that reads from a Kafka topic
▪ How to configure a Flume sink that publishes to a Kafka topic
Chapter Topics

Integrating Flume and Kafka

- Overview
- Use Cases
- Configuration
- Tips for Deployment
- Essential Points
- Hands-On Exercise: Using Kafka as a Flume Sink
- Hands-On Exercise: Using Kafka as a Flume Source
Should I Use Kafka or Flume?

- Both Flume and Kafka are widely used for data ingest
  - Although these tools differ, their functionality has some overlap
  - Some use cases could be implemented with *either* Flume or Kafka

- How do you determine which is a better choice for *your* use case?
Characteristics of Flume

- **Flume is efficient at moving data from a single source into Hadoop**
  - It offers sinks that write to HDFS, an HBase table, or a Solr index
  - Easily configured to support common scenarios, without writing code
  - Can also process and transform data during the ingest process
Characteristics of Kafka

- Kafka is a publish-subscribe messaging system
  - It offers more flexibility for connecting multiple systems
  - Provides better durability and fault tolerance than Flume
  - Typically requires writing code for producers and/or consumers
  - No direct support for processing messages or loading into Hadoop
Flafka = Flume + Kafka

- Both systems have strengths and limitations
- You don’t necessarily have to choose between them
  - It is possible to use both when implementing your use case
- Flafka is the informal name for Flume-Kafka integration
  - It uses a Flume agent to read from or write messages to Kafka
- It is implemented as a Kafka source and sink for Flume
  - These components ship with Flume, starting with CDH 5.2.0
  - A Kafka channel also now ships with Flume, starting with CDH 5.3.0
Chapter Topics

Integrating Flume and Kafka

- Overview
- Use Cases
- Configuration
- Tips for Deployment
- Essential Points
- Hands-On Exercise: Using Kafka as a Flume Sink
- Hands-On Exercise: Using Kafka as a Flume Source
Using Flume as a Kafka Producer

- By using the Kafka sink, Flume can publish messages to a topic
- In this example, an application uses Flume to publish application events
  - The application sends data to the Flume source when events occur
  - The event data is buffered in the channel until it is taken by the sink
  - Since we use a Kafka sink, the events are published to a specified topic
  - Any Kafka consumer can then read messages for application events
Using Flume as a Kafka Consumer

- By using the Kafka source, Flume can read messages from a topic
  - It can then write them to your destination of choice using a Flume sink

- In this example, the Producer sends messages to Kafka brokers
  - The Flume agent uses a Kafka source, which acts as a consumer
  - The Kafka source reads messages in a specified topic
  - The message data is buffered in the channel until it is taken by the sink
  - The sink then writes the data into HDFS
Chapter Topics

Integrating Flume and Kafka

- Overview
- Use Cases
- Configuration
- Tips for Deployment
- Essential Points
- Hands-On Exercise: Using Kafka as a Flume Sink
- Hands-On Exercise: Using Kafka as a Flume Source
The table below describes several properties of the Kafka sink

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>Must be set to org.apache.flume.sink.kafka.KafkaSink</td>
</tr>
<tr>
<td>brokerList</td>
<td>Comma-separated list of brokers (format host:port) to contact</td>
</tr>
<tr>
<td>topic</td>
<td>The topic in Kafka to which the messages will be published.</td>
</tr>
<tr>
<td>batchSize</td>
<td>How many messages to process in one batch</td>
</tr>
</tbody>
</table>

![Application Flume Agent Kafka Cluster Diagram]
This is the Flume configuration for the example on the previous slide

```
# Define names for the source, channel, and sink
agent1.sources  = source1
agent1.channels = channel1
agent1.sinks = sink1

# Define the properties of our source, which receives event data
agent1.sources.source1.type = netcat
agent1.sources.source1.bind = localhost
agent1.sources.source1.port = 44444
agent1.sources.source1.channels = channel1

# Define the properties of our channel
agent1.channels.channel1.type = memory
agent1.channels.channel1.capacity = 10000
agent1.channels.channel1.transactionCapacity = 1000
```

Note: file continues on next slide
Configuration: Using Flume as a Kafka Producer (2)

- The remaining portion of the configuration file sets up the Kafka sink

```
# Define our Kafka sink, which publishes to the app_event topic
agent1.sinks.sink1.type = org.apache.flume.sink.kafka.KafkaSink
agent1.sinks.sink1.topic = app_events
agent1.sinks.sink1.brokerList = localhost:9092
agent1.sinks.sink1.batchSize = 20
agent1.sinks.sink1.channel = channel1
```
Configuration: Using Flume as a Kafka Consumer (1)

- The table below describes several properties of the Kafka source

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>org.apache.flume.source.kafka.KafkaSource</td>
</tr>
<tr>
<td>zookeeperConnect</td>
<td>ZooKeeper connection string (e.g., localhost:2181)</td>
</tr>
<tr>
<td>groupId</td>
<td>Unique ID to use for the consumer group (default: flume)</td>
</tr>
<tr>
<td>topic</td>
<td>Name of Kafka topic from which messages will be read</td>
</tr>
</tbody>
</table>
This is the Flume configuration for the example on the previous slide

- It defines a source for reading messages from a Kafka topic

```java
# Define names for the source, channel, and sink
agent1.sources = source1
agent1.channels = channel1
agent1.sinks = sink1

# Define a Kafka source that reads from the calls_placed topic
# The "type" property line wraps around due to its long value
agent1.sources.source1.type = org.apache.flume.source.kafka.KafkaSource
agent1.sources.source1.zookeeperConnect = localhost:2181
agent1.sources.source1.topic = calls_placed
agent1.sources.source1.groupId = flume
agent1.sources.source1.channels = channel1
```

Note: file continues on next slide
This is the Flume configuration for the example on the previous slide

```bash
# Define the properties of our channel
agent1.channels.channel1.type = memory
agent1.channels.channel1.capacity = 10000
agent1.channels.channel1.transactionCapacity = 1000

# Define the sink that writes call data to HDFS
agent1.sinks.sink1.type=hdfs
agent1.sinks.sink1.hdfs.path = /user/training/calls_placed
agent1.sinks.sink1.hdfs.fileType = DataStream
agent1.sinks.sink1.hdfs.fileSuffix = .csv
agent1.sinks.sink1.channel = channel1
```
Chapter Topics

Integrating Flume and Kafka

- Overview
- Use Cases
- Configuration
- Tips for Deployment
- Essential Points
- Hands-On Exercise: Using Kafka as a Flume Sink
- Hands-On Exercise: Using Kafka as a Flume Source
Use Kafka for Custom Producers and Consumers

- **Kafka has a significantly smaller producer and consumer ecosystem**
  - Use Kafka if you’re prepared to implement producers and consumers

- **Use Flume if its sources and sinks match your requirements**
  - Flume has many built-in sources and sinks from which to choose
  - Using them requires only configuration, not writing code
Use Flume for Filtering and Transforming Data

- **Flume can process data in-flight using interceptors**
  - These can be very useful for filtering or transforming data

- **Kafka requires an external stream processing system**
  - Spark Streaming is a popular choice
Use Kafka for High Availability

- Both Kafka and Flume are reliable systems that can guarantee no data loss
- However, Flume does not replicate events
  - As a result, if a node with the Flume agent crashes, you will lose access to the events in the channel until you recover the disks
  - This is true even when using the file channel
- Use Kafka if you need an ingest pipeline with very high availability
A Flume Agent Can Write to Multiple Sinks

- You can configure a Flume agent to use multiple channels
  - Each channel sends data to an associated sink
- This can be used to write data to HDFS and Kafka simultaneously
Chapter Topics

Integrating Flume and Kafka

- Overview
- Use Cases
- Configuration
- Tips for Deployment

- Essential Points
  - Hands-On Exercise: Using Kafka as a Flume Sink
  - Hands-On Exercise: Using Kafka as a Flume Source
Essential Points

- Flume and Kafka are distinct systems with different designs
  - You must weight the advantages and disadvantages of each when selecting the best tool for your use case

- Flume and Kafka can be combined with Flafka
  - This is the informal name for Flume components for Kafka integration
  - You can read messages from a topic using a Kafka source
  - You can publish messages to a topic using a Kafka sink
Bibliography

The following offer more information on topics discussed in this chapter

- **Flafka: Apache Flume Meets Apache Kafka for Event Processing**
  - [http://tiny.cloudera.com/kmc02a](http://tiny.cloudera.com/kmc02a)

- **Designing Fraud-Detection Architecture That Works Like Your Brain Does**
  - [http://tiny.cloudera.com/kmc02b](http://tiny.cloudera.com/kmc02b)
Chapter Topics

Integrating Flume and Kafka

- Overview
- Use Cases
- Configuration
- Tips for Deployment
- Essential Points
- Hands-On Exercise: Using Kafka as a Flume Sink
- Hands-On Exercise: Using Kafka as a Flume Source
Hands-On Exercise: Using Kafka as a Flume Sink (Flafka)

- In this exercise, you will use Flume’s Kafka sink to write data that was received by a Flume agent into a Kafka sink
  - Please refer to the Hands-On Exercise Manual for instructions
Chapter Topics

Integrating Flume and Kafka

- Overview
- Use Cases
- Configuration
- Tips for Deployment
- Essential Points
- Hands-On Exercise: Using Kafka as a Flume Sink
- Hands-On Exercise: Using Kafka as a Flume Source
Hands-On Exercise: Using Kafka as a Flume Source (Flafka)

- In this exercise, you will use Flume’s Kafka source to read data published to a Kafka topic and write it to a directory in HDFS
  - Please refer to the Hands-On Exercise Manual for instructions