Data Schools

Module 4:
Introduction To Kafka

Introduction

Agenda

- Introduction - Messaging Basics
- Kafka – Architecture
- Kafka – Partitioning & Topics
- Summary

Agenda

- Introduction - Messaging Basics
- Kafka – Architecture
- Kafka – Partitioning & Topics
- Summary

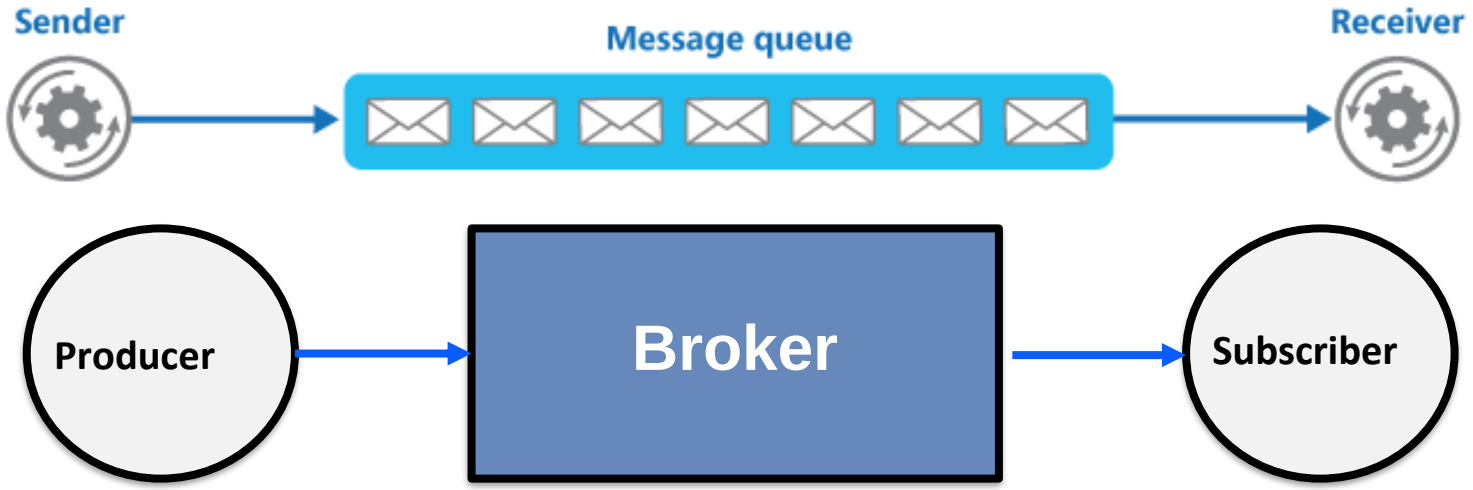
Introduction

- When used in the right way and for the right use case, Kafka has unique attributes that make it a highly attractive option for data integration.

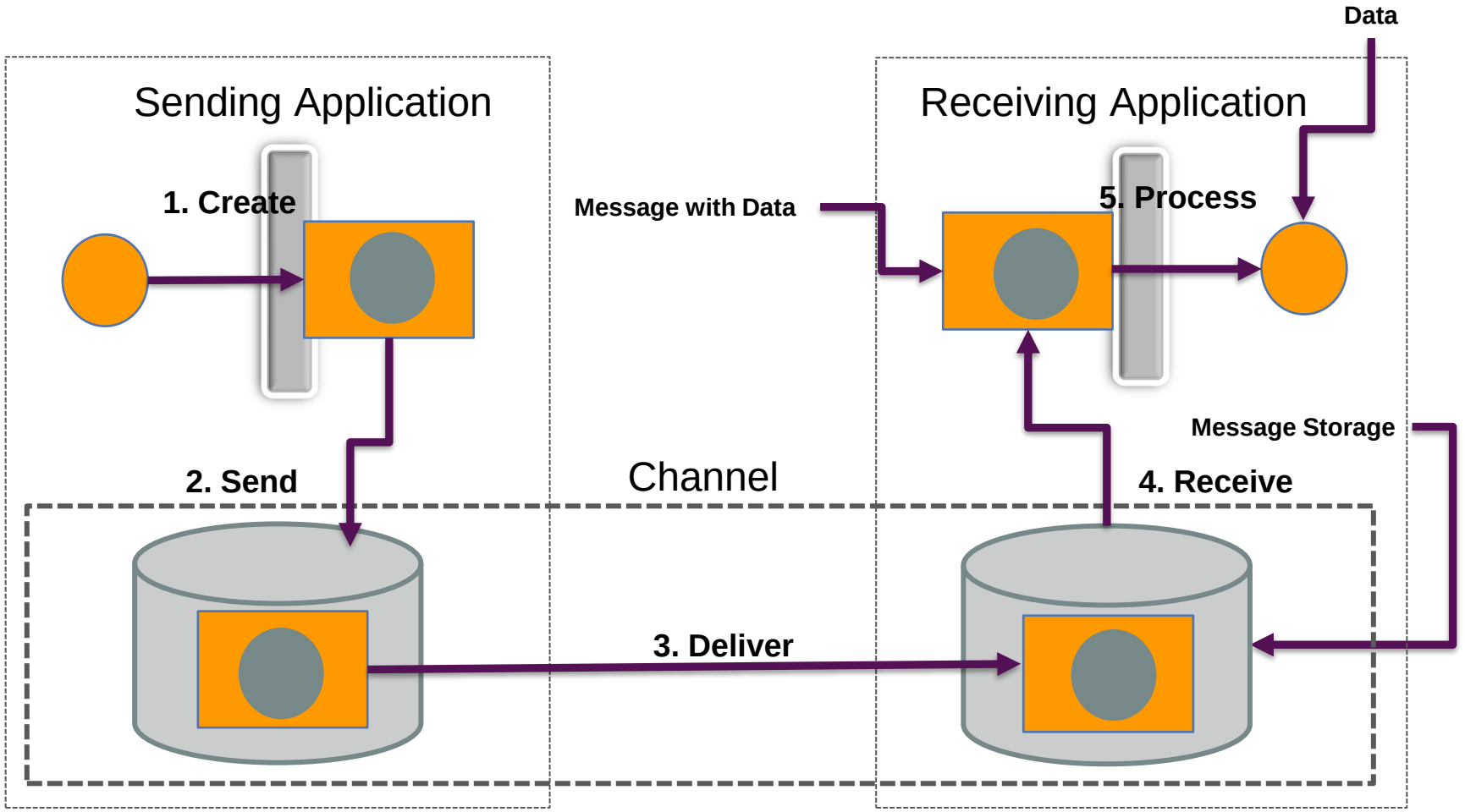
- Data Integration is the combination of technical and business processes used to combine data from disparate sources into meaningful and valuable information.
- A complete data integration solution encompasses discovery, cleansing, monitoring, transforming and delivery of data from a variety of sources
- Messaging is a key data integration strategy employed in many distributed environments such as the cloud.
- Messaging supports asynchronous operations, enabling you to decouple a process that consumes a service from the process that implements the service.

Messaging Architectures: What is Messaging?

- Application-to-application communication
- Supports asynchronous operations.
- Message:
  - A message is a self-contained package of data and network routing headers.
- Broker:
  - Intermediary program that translates messages from the formal messaging protocol of the publisher to the formal messaging protocol of the receiver.



Messaging Basics



Steps to Send a Message

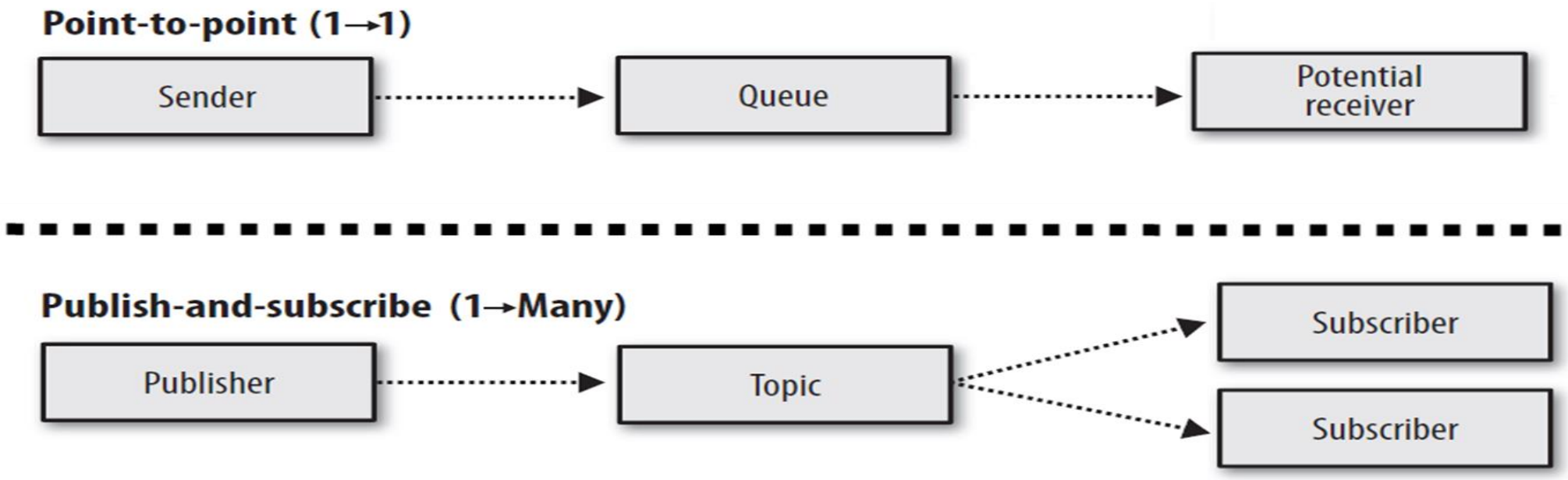
Reference: Enterprise Integration Patterns - Gregor Hohpe and Bobby Woolf

Agenda

- Introduction - Messaging Basics
- Kafka – Architecture
- Kafka – Partitioning & Topics
- Summary

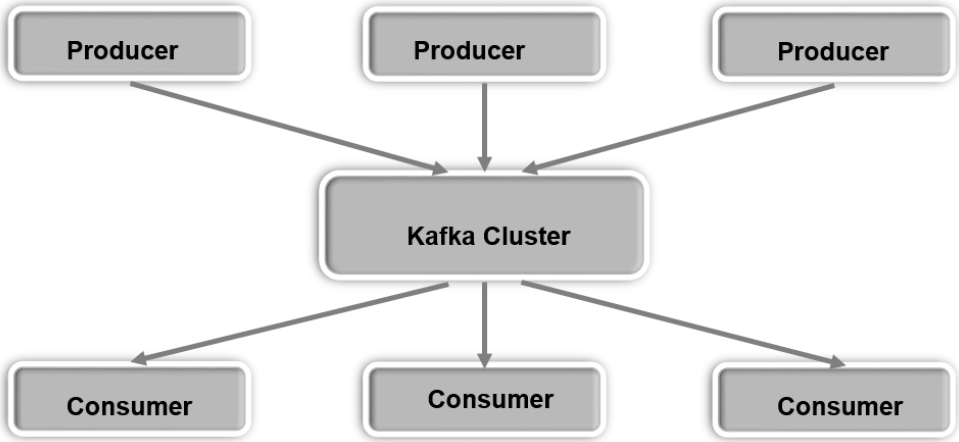
Messaging Architectures: Messaging Models

1. Point to Point
2. Publish and Subscribe



Kafka is an example of publish-and-subscribe messaging model

Kafka Overview



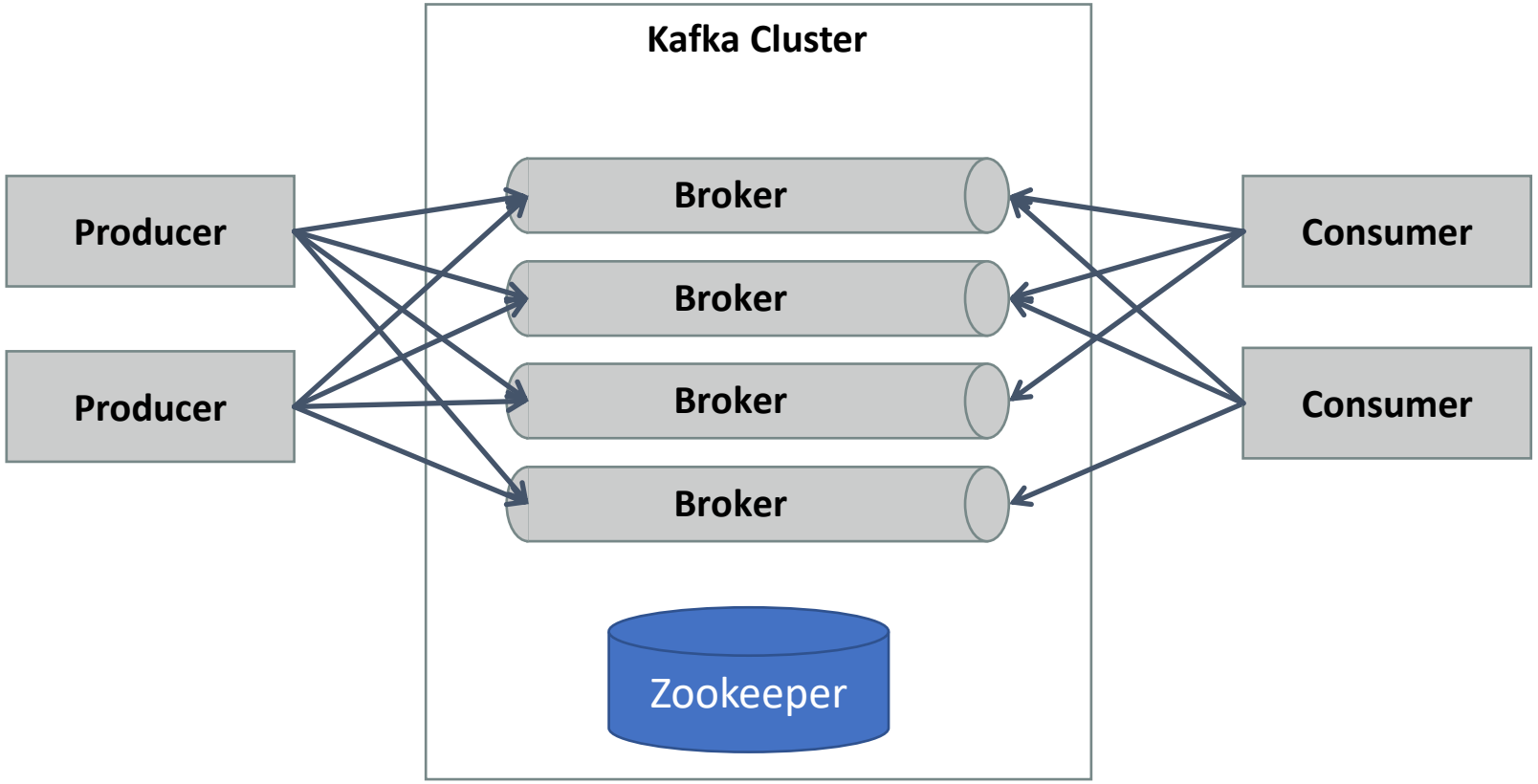
- Kafka is a unique distributed publish-subscribe messaging system written in the Scala language with multi-language support and runs on the Java Virtual Machine (JVM).
- Kafka relies on another service named Zookeeper – a distributed coordination system – to function.
- Kafka has high-throughput and is built to scale-out in a distributed model on multiple servers.
- Kafka persists messages on disk and can be used for batched consumption as well as real time applications.

Key Terminology

- Kafka maintains feeds of messages in categories called topics.
- Processes that publish messages to a Kafka topic are called producers.
- Processes that subscribe to topics and process the feed of published messages are called consumers.
- Kafka is run as a cluster comprised of one or more servers each of which is called a broker.

- Communication between all components is done via a high performance simple binary API over TCP protocol

Kafka Architecture

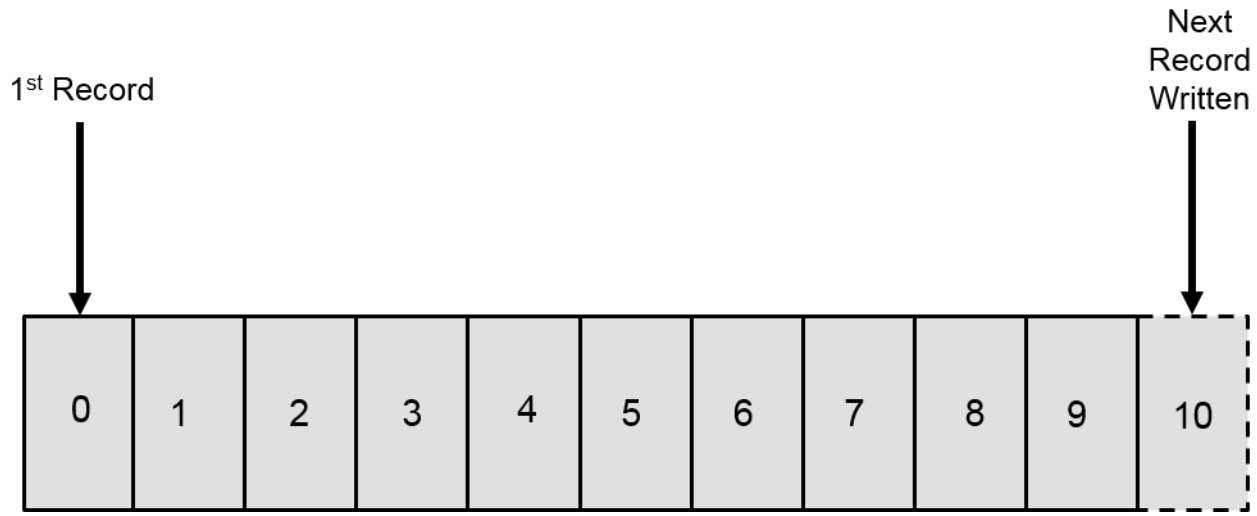


Agenda

- Introduction - Messaging Basics
- Kafka – Architecture
- Kafka – Partitioning & Topics
- Summary

Understanding Kafka

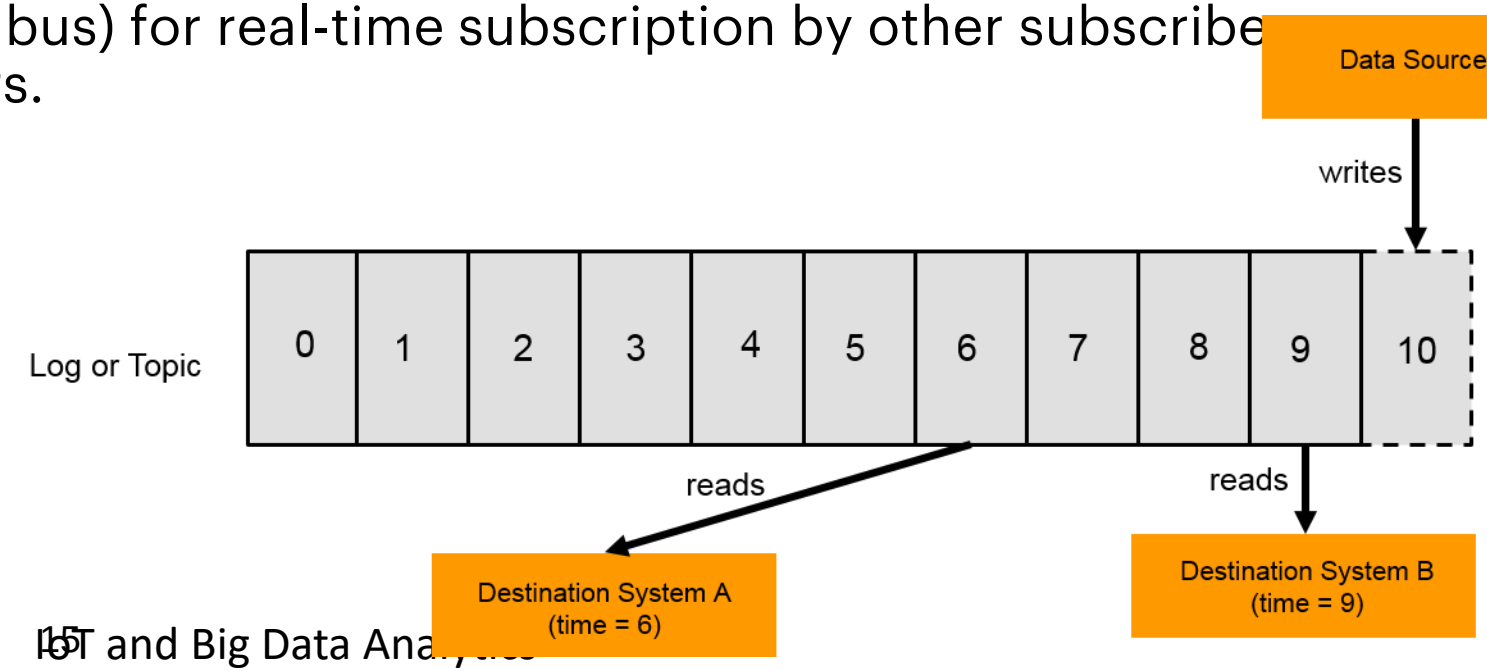
- Kafka is based on the simple storage-abstraction concept called a log, an append-only totally-ordered sequence of records ordered by time.



- Records are appended to the end of the record and reads proceed from left to right in the log (or topic).
- Each entry is assigned a unique sequential log-entry number (an offset).
- The log entry number is a convenient property that correlates to the notion of a “timestamp” entry but is decoupled from any clock due to the distributed nature of Kafka.

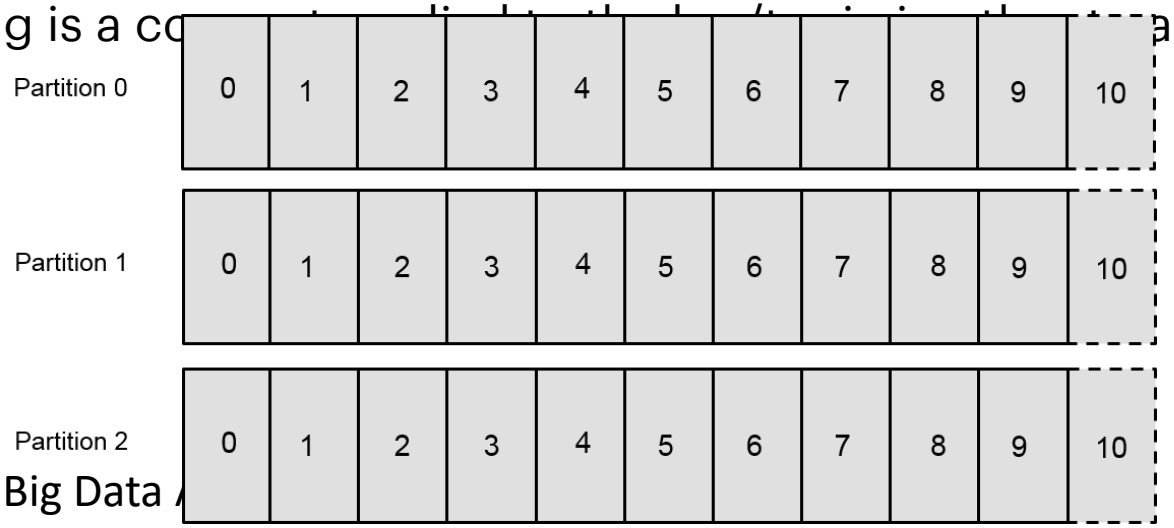
Kafka Key Design Concepts

- A log is synonymous to a file or table where the records are appended and sorted by the concept of time.
- Conceptually, the log is a natural data-structure for handling data-flow between systems.
- Kafka is designed for centralizing an organization's data into an enterprise log (message bus) for real-time subscription by other subscribe ation consumers.



Kafka Conceptual Design

- Each logical data source can be modeled as a log corresponding to a topic or data feed in Kafka.
- Each subscribing consuming application should read as quickly as it can from each topic, persist the record it reads into it's own data store and advances the offset to the next message entry to be read.
- Subscribers can be any type of data system or middleware system like a cache, Hadoop, a streaming system like Spark or Storm, a search system, a web services provisioning system, a data warehouse, etc.
- In Kafka, partitioning is a co[illegible]allow horizontal scaling.



Kafka Logical Design

- Each partition is a totally ordered log within a topic, and there is no global ordering between partitions.
- Assignment of messages to specific partitions is controlled by the publisher and may be assigned based on a unique identification key or messages can be allowed to be randomly assigned to partitions.
- Partitioning allows throughput to scale linearly with the Kafka cluster size.

Topics - Introduction

- When adopting a streaming platform such as Apache Kafka, some important questions to answer are:

- What topics are you going to use?
- If you have a bunch of different events that you want to publish to Kafka as messages, do you put them in the same topic, or do you split them across different topics?
- Two extremes – one topic or millions?

Topics – General Principle

- Topic = collection of events of the same type?
  - put all events of the same type in the same topic,
  - and use different topics for different event types.

- What about cases where the ordering of events matters?

Topics – Ordering Problems

- Ordering is not preserved across partitions

- Every topic has at least 1 partition
- If the order of event are important, having these events in different topics = different partitions

Topics – When to Split, When to Combine

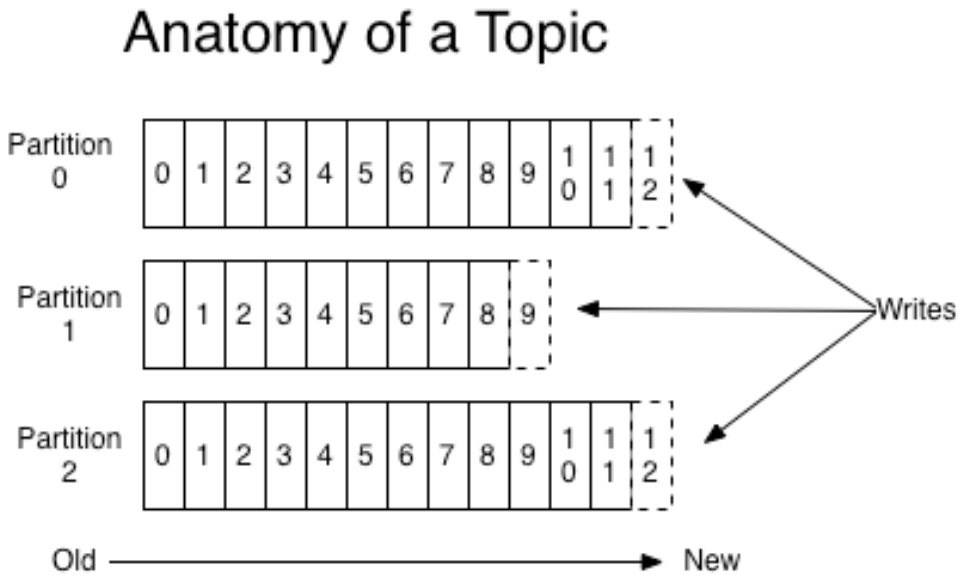
1. The most important rule is that any events that need to stay in a fixed order must go in the same topic
2. If an event entity depends on another, they should be in the same topic. If unrelated the separate
3. Try not to split up events with multiple entities initially
4. If several consumers all read a particular group of topics, this suggests that maybe those topics should be combined

Kafka Topics

- Kafka topics should have a small number of consumer groups assigned with each one representing a “logical subscriber”.
- Kafka topic consumption can be scaled by increasing the number of consumer subscriber instances within the same group which will automatically load-balance message consumption.
- Kafka has a notion of partitioning within a topic to provide the notion of parallel consumption
- Partitions in a topic are assigned to the consumers within a consumer group.
- There can be no more consumer instances within a consumer group than partitions within a topic.
- If the total order in which messages are published is important in the consumption, then a single partition for the topic is the solution which will mean only one consumer process in the consumer group.

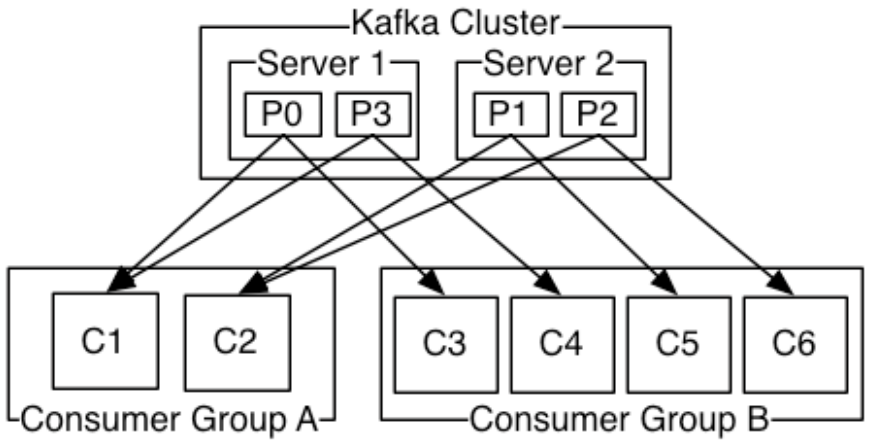
Kafka Topic Partitions

- A topic consists of partitions.
- Partition: ordered + immutable sequence of messages that is continually appended to



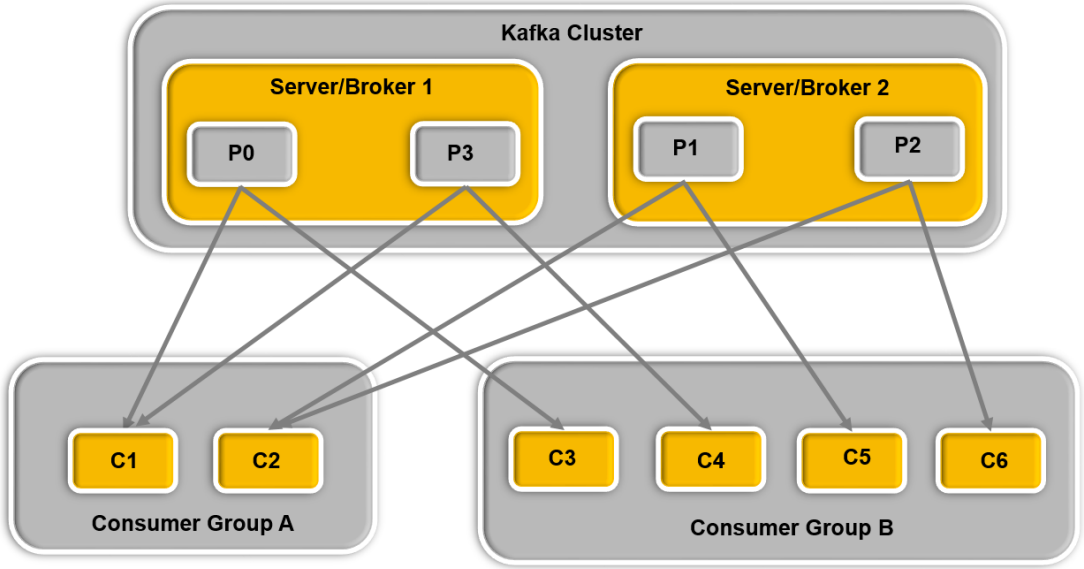
Kafka Topic Partitions

- #partitions of a topic is configurable
- #partitions determines max consumer (group) parallelism
- Cf. parallelism of Storm's KafkaSpout via builder.setSpout(,,N)



- Consumer group A, with 2 consumers, reads from a 4-partition topic
- Consumer group B, with 4 consumers, reads from the same topic

Kafka Consumer Groups



Two Broker/Server Kafka Cluster with four partitions (P0-P3) serving two consumer groups.
Consumer Group A has two instances and Consumer Group B has four instances of consumers.

- Kafka assigns the partitions in a topic to the consumer instances in a consumer group to provide ordering guarantees and load balancing over a pool of consumer process. Note that there can be no more consumer instances per group than total partition count.

Kafka Environment Properties

- Ensure you have access to downloading libraries from the web.
- Have at least 15 GB of free hard disk space on your local machine.
- Have at least 8GB (preferably 16GB) of RAM on your local machine.
- Have a JRE of version 1.7 and above installed on the local machine.
- Download and install Eclipse Mars (or the current release) on your local machine.
- Download and install VMware player for Windows on the local machine
- Download and install Git from the URL https://git-scm.com/
- Download and install Maven https://maven.apache.org/download.cgi
- Download the latest stable version of Gradle http://gradle.org/gradle-download/
- Download Scala (use the Scala version compatible with the Kafka download Scala version – in this document Scala version 2.10 is utilized)
- Make sure all the necessary command paths for Git, Maven, Gradle, etc are in the Windows Environment and Path.

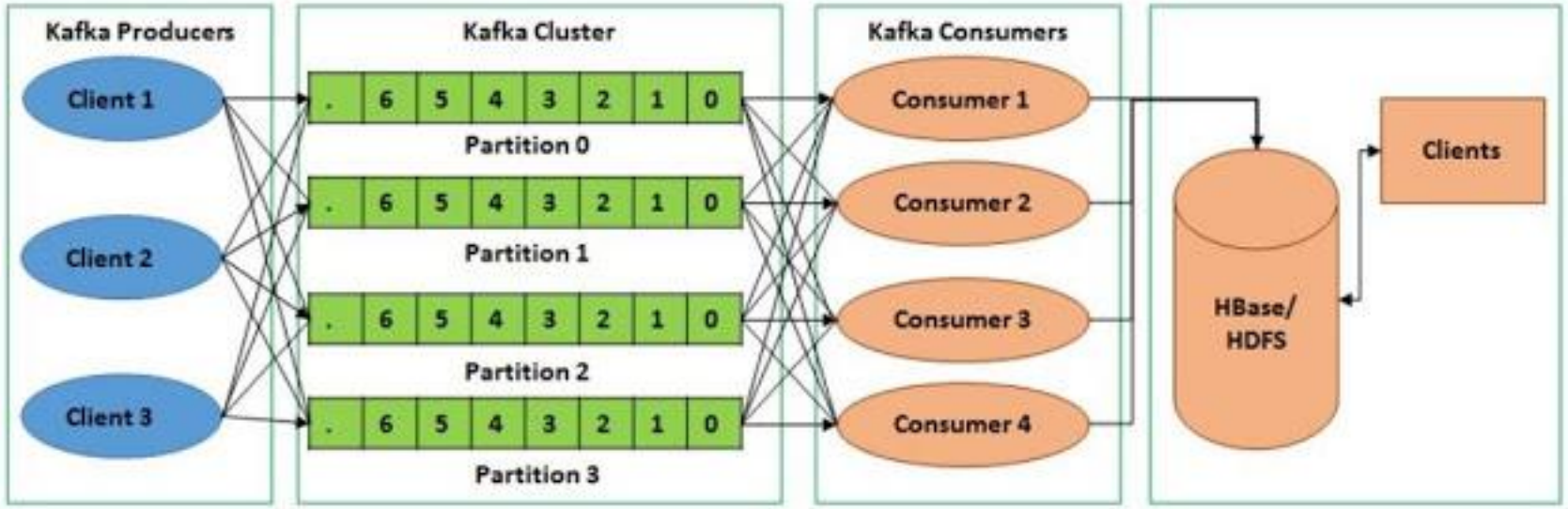
Kafka Environment Setup

- The Kafka environment can be set up on a local machine in Windows, Linux or in a virtual environment on the local machine.
- Go to the Kafka Download URL: https://kafka.apache.org/downloads.html
  - The current Kafka download site has current release and previous release versions of Kafka with there corresponding Scala version binary downloads.
  - The download releases have a suffix of *.tgz which means the binaries are gzipd compiled as Linux tar balls.
  - To get the Windows binaries, the source code needs to be downloaded and compiled on Windows.

Partitions

- More Partitions Lead to Higher Throughput
  - topic partition is the unit of parallelism in Kafka
- More Partitions Requires More Open File Handles
- More Partitions May Increase Unavailability
- More Partitions May Require More Memory In the Client

Partitions



- The more partitions the greater the Zookeeper overhead
  - With large partition numbers ensure proper ZK capacity
- Message ordering can become complex
  - Single partition for global ordering
  - Consumer-handling for ordering
- The more the partitions the longer the leader fail-over time

Agenda

- Introduction - Messaging Basics
- Kafka – Architecture
- Kafka – Partitioning & Topics
- Summary

Summary

- When used in the right way and for the right use case, Kafka has unique attributes that make it a highly attractive option for data integration.
- Kafka is a unique distributed publish-subscribe messaging system written in the Scala language with multi-language support and runs on the Java Virtual Machine (JVM).