cloudera

Cloudera Custom Training for Kafka

190104

cloudera®

Introduction

Chapter 1

cloudera®

Course Chapters

- **Introduction**
- Kafka Overview
- Deploying Apache Kafka
- Kafka Command Line Basics
- Kafka Java API Basics
- Improving Availability through Replication
- Improving Application Reliability
- Improving Application Scalability
- Monitoring and Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka Cluster
- Replicating Kafka to Another Cluster
- Conclusion

Trademark Information

- **The names and logos of Apache products mentioned in Cloudera training courses, including those listed below, are trademarks of the Apache Software Foundation**

Apache Accumulo
Apache Avro
Apache Bigtop
Apache Crunch
Apache Flume
Apache Hadoop
Apache HBase
Apache HCatalog
Apache Hive
Apache Impala
Apache Kafka
Apache Kudu
Apache Lucene
Apache Mahout
Apache Oozie
Apache Parquet
Apache Pig
Apache Sentry
Apache Solr
Apache Spark
Apache Sqoop
Apache Tika
Apache Whirr
Apache ZooKeeper

- **All other product names, logos, and brands cited herein are the property of their respective owners**

Chapter Topics

Introduction

- **About This Course**
- About Cloudera
- Course Logistics
- Introductions
- About the Exercise Environment

Course Objectives (1)

During this course, you will learn

- **What Apache Kafka is and how organizations are using it**
- **Which roles producers, consumers, and brokers play in Kafka**
- **How Cloudera's distribution of Kafka relates to Apache Kafka**
- **What to consider when planning a Kafka installation**
- **How to deploy a typical Kafka cluster using Cloudera Manager**
- **How to manage topics from both the command line and custom Java code**
- **How to produce and consume messages from the command line and custom Java code**
- **How to increase fault tolerance through replication**

Course Objectives (2)

- How several configuration properties affect message delivery and storage
- How messages are partitioned for scalability and how it affects clients
- Which metrics are most important to monitor
- How to troubleshoot common problems and performance issues
- How to improve efficiency by using compression and custom serialization
- Which technologies Cloudera recommends for securing Kafka
- How you can mirror Kafka data to another cluster

Chapter Topics

Introduction

- About This Course
- **About Cloudera**
- Course Logistics
- Introductions
- About the Exercise Environment

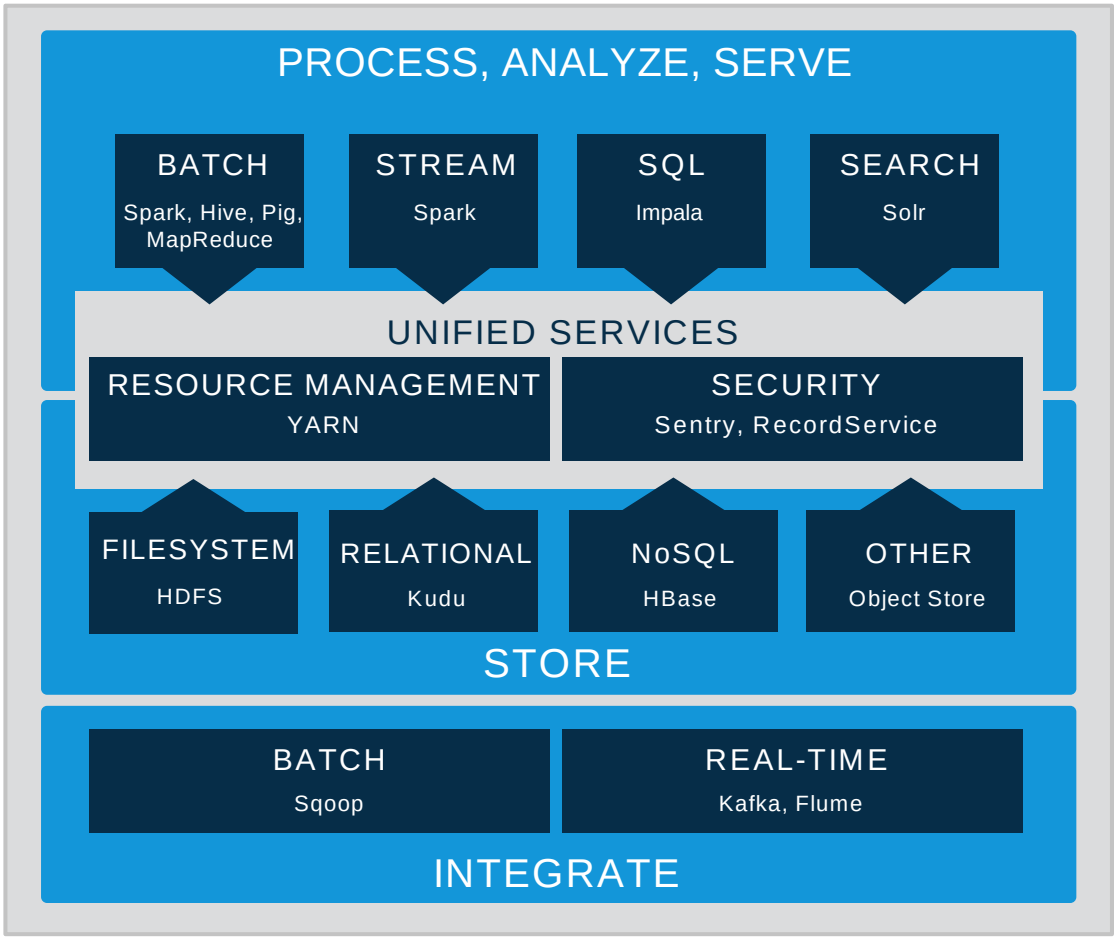
About Cloudera

- The leader in Apache Hadoop-based software and services
- Founded by Hadoop experts from Facebook, Yahoo, Google, and Oracle
- Provides support, consulting, training, and certification for Hadoop users
- Staff includes committers to virtually all Hadoop projects
- Many authors of authoritative books on Apache Hadoop projects

CDH

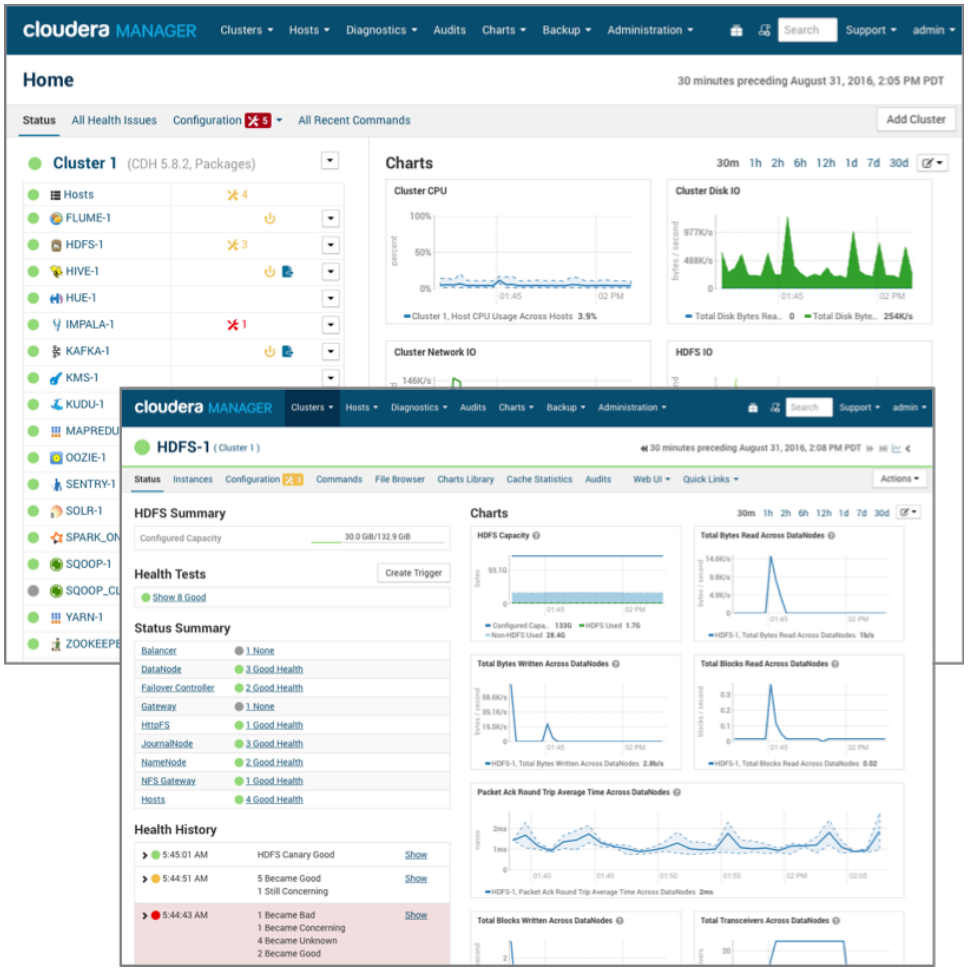
CDH (Cloudera's Distribution including Apache Hadoop)

- 100% open source, enterprise-ready distribution of Hadoop and related projects
- The most complete, tested, and widely deployed distribution of Hadoop
- Integrates all the key Hadoop ecosystem projects
- Available as RPMs and Ubuntu, Debian, or SuSE packages, or as a tarball

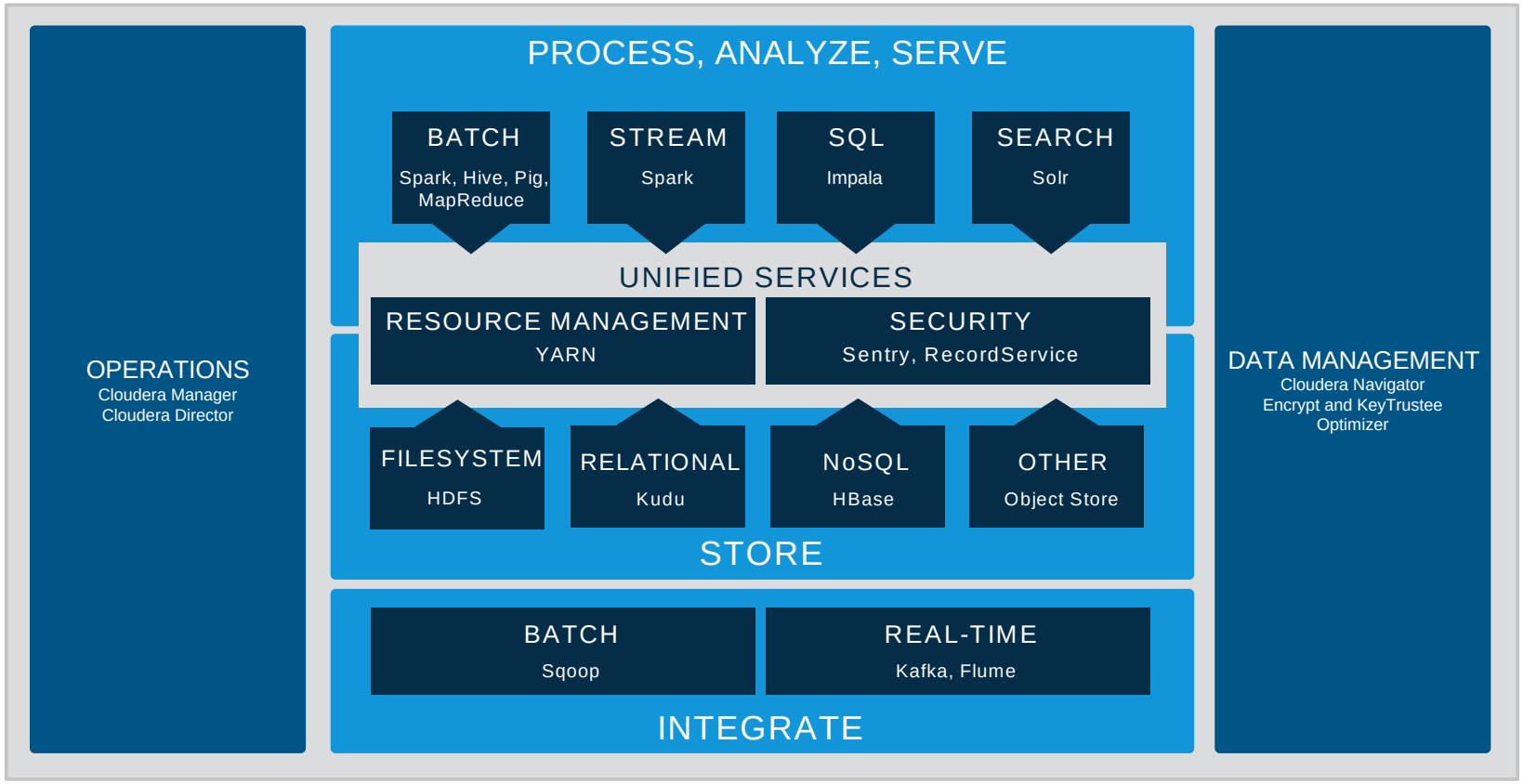


Cloudera Express

- **Cloudera Express**
  - Completely free to download and use
- **The best way to get started with Hadoop**
- **Includes CDH**
- **Includes Cloudera Manager**
  - End-to-end administration for Hadoop
  - Deploy, manage, and monitor your cluster

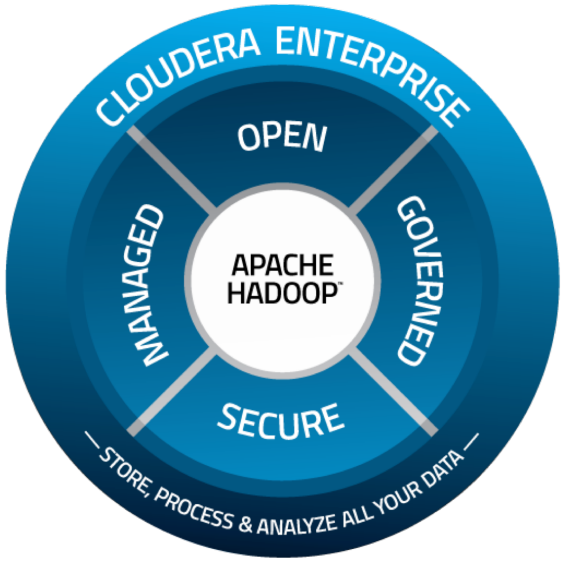


Cloudera Enterprise (1)



Cloudera Enterprise (2)

- **Subscription product including CDH and Cloudera Manager**
- **Provides advanced features, such as**
  - Operational and utilization reporting
  - Configuration history and rollbacks
  - Rolling updates and service restarts
  - External authentication (LDAP/SAML)*
  - Automated backup and disaster recovery
- **Specific editions offer additional capabilities, such as**
  - Governance and data management (Cloudera Navigator)
  - Active data optimization (Cloudera Navigator Optimizer)
  - Comprehensive encryption (Cloudera Navigator Encrypt)
  - Key management (Cloudera Navigator Key Trustee)



* **LDAP** = Lightweight Directory Access Protocol and **SAML** = Security Assertion Markup Language

Cloudera University

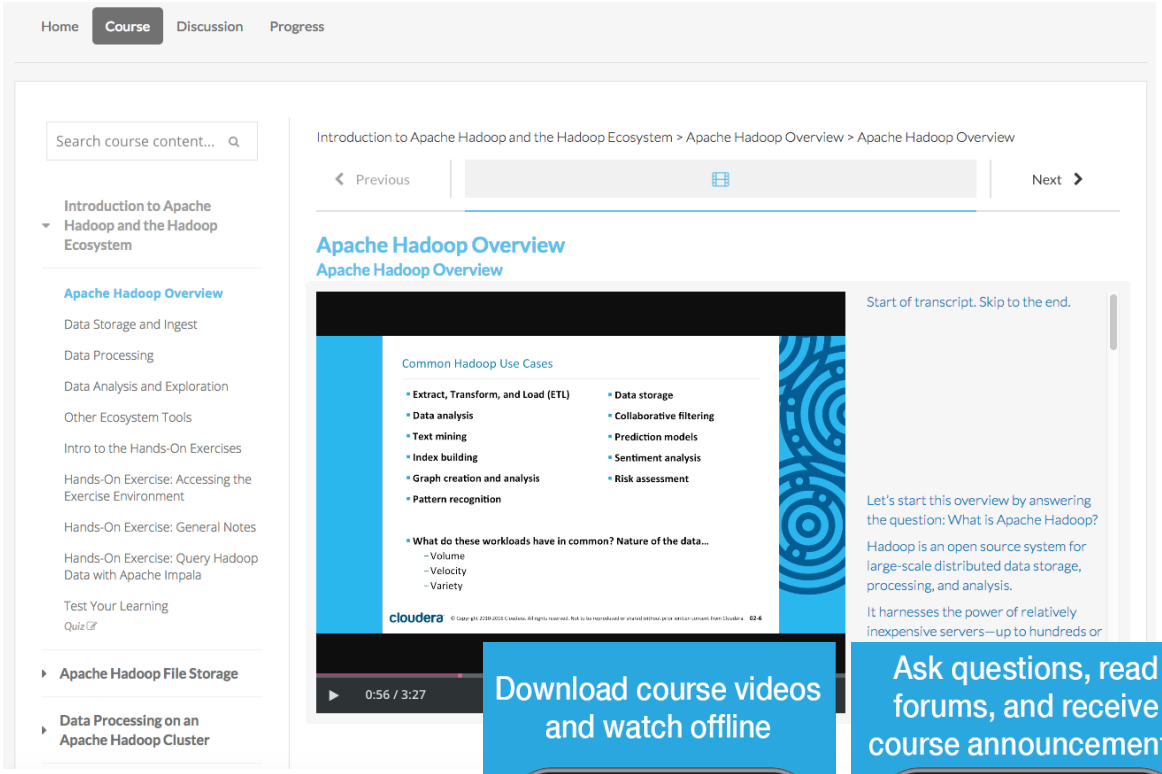
- **The leader in Apache Hadoop-based training**
- **We offer a variety of courses, both instructor-led (in physical or virtual classrooms) and self-paced OnDemand, including:**
  - *Developer Training for Apache Spark and Hadoop*
  - *Cloudera Administrator Training for Apache Hadoop*
  - *Cloudera Data Analyst Training*
  - *Cloudera Search Training*
  - *Cloudera Data Scientist Training*
  - *Cloudera Training for Apache HBase*
- **We also offer private courses:**
  - Can be delivered on-site, virtually, or online OnDemand
  - Can be tailored to suit customer needs

Cloudera University OnDemand

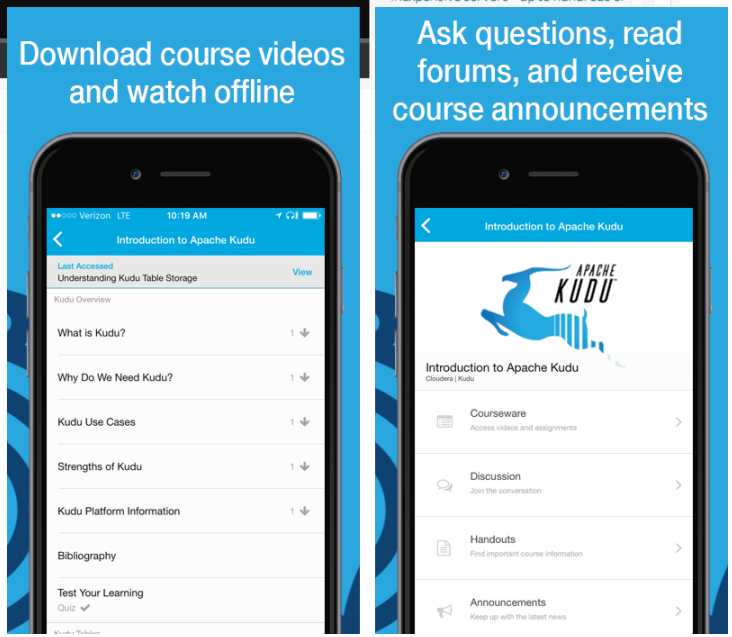
- **Our OnDemand platform includes**
  - A full catalog of courses, with free updates
  - Exclusive OnDemand-only course: *Cloudera Security Training*
  - Free courses such as *Essentials* and *Cloudera Director*
  - OnDemand-only modules such as *Cloudera Director* and *Cloudera Navigator*
  - Searchable content within and across courses
- **Courses include:**
  - Video lectures and demonstrations with searchable transcripts
  - Hands-on exercises through a browser-based virtual cluster environment
  - Discussion forums monitored by Cloudera course instructors
- **Purchase access to a library of courses or individual courses**
- **See the Cloudera OnDemand information page for more details or to make a purchase, or go directly to the OnDemand Course Catalog**

Accessing Cloudera OnDemand

- **Cloudera OnDemand subscribers can access their courses online through a web browser**



- **Cloudera OnDemand is also available through an iOS app**
  - Search for "Cloudera OnDemand" in the iOS App Store
  - iPad users: change your Store search settings to "iPhone only"



Cloudera Certification

- **The leader in Apache Hadoop-based certification**
- **All Cloudera professional certifications are hands-on, performance-based exams requiring you to complete a set of real-world tasks on a working multi-node CDH cluster**
- **We offer two levels of certifications**
  - **Cloudera Certified Professional (CCP)**
    - The industry's most demanding performance-based certification, CCP Data Engineer evaluates and recognizes your mastery of the technical skills most sought after by employers
    - CCP Data Engineer
  - **Cloudera Certified Associate (CCA)**
    - To achieve CCA certification, you complete a set of core tasks on a working CDH cluster instead of guessing at multiple-choice questions
    - CCA Spark and Hadoop Developer
    - CCA Data Analyst
    - CCA Administrator

Chapter Topics

Introduction

- About This Course
- About Cloudera
- **Course Logistics**
- Introductions
- About the Exercise Environment

Logistics

- Class start and finish time
- Lunch
- Breaks
- Restrooms
- Wi-Fi access
- Virtual machines

Your instructor will give you details on how to access the course materials for the class

Chapter Topics

Introduction

- About This Course
- About Cloudera
- Course Logistics
- **Introductions**
- About the Exercise Environment

Introductions

- **About your instructor**
- **About you**
  - Where do you work? What do you do there?
  - What programming languages have you used?
  - Do you have experience with UNIX or Linux?
  - How much Hadoop experience do you have?
  - What do you expect to gain from this course?

Chapter Topics

Introduction

- About This Course
- About Cloudera
- Course Logistics
- Introductions
- **About the Exercise Environment**

Scenario Explanation

- **Hands-On Exercises let you practice what you have learned**
- **These exercises are based on a hypothetical scenario**
  - However, the concepts apply to nearly any organization
- **Loudacre Mobile is a fast-growing wireless carrier**
  - Founded in 2008 and headquartered in Palo Alto, California
  - Now provides service to 130,000 customers throughout the western USA

Exercise Environment Overview

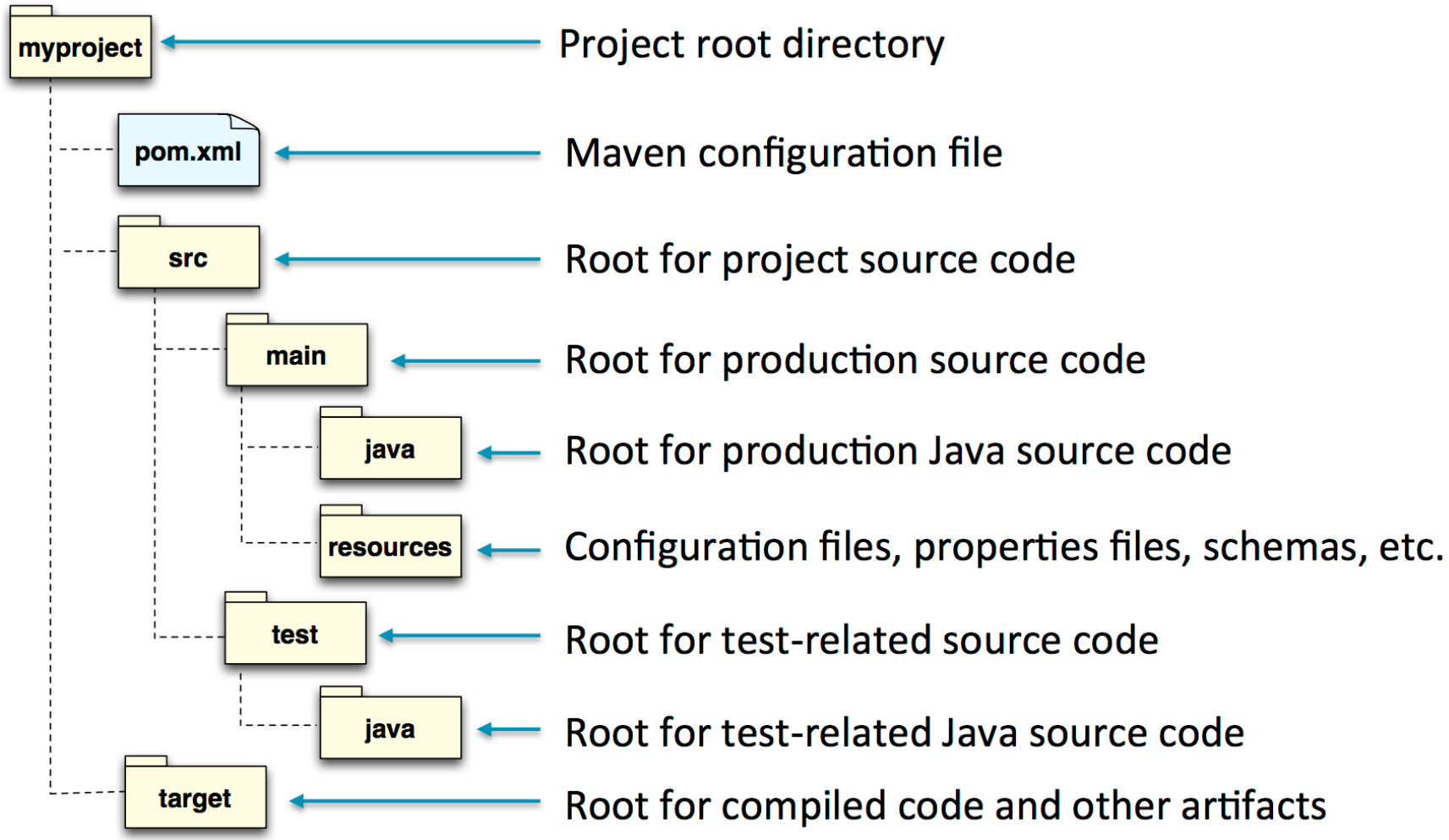
- **Each student has a five-node cluster running in the AWS cloud**
  - Managed by Cloudera Manager
  - Versions used
    - Cloudera Manager
    - CDH
    - Kafka
    - JDK
- **Your instructor will explain more about the cluster**
  - When it will be available
  - How to access it
  - Which setup tasks you must perform

Development Environment

- **Course uses the same tools engineers typically use for development**
  - IDE (Eclipse)
  - Unit testing library (JUnit)
  - Build management tool (Maven)

Maven Project Layout

- **Standard Java project layout is based on "convention over configuration"**



Maven Goals

- **Build tasks are accomplished by invoking one or more *goals***
- **Goals are specified as arguments to the `mvn` command**
  - The `compile` goal will compile the project's source code

```
$ mvn compile
```

- **You may specify multiple goals at once**
  - This example deletes transient artifacts and compiles project sources

```
$ mvn clean compile
```

- **The `package` goal creates a JAR after compiling and testing the code**

```
$ mvn package
```

Using Maven to Run Java Code

- **Maven's `java:exec` goal enables you to run the code you built**
  - The classpath is configured automatically
  - Specify the class to run with the `exec.mainClass` system property
  - Specify program arguments with the `exec.args` system property
- **This example passes four arguments to the program's `main` method**

```
$ mvn exec:java \
    -Dexec.mainClass="com.cloudera.example.PrintLocalTime" \
    -Dexec.args="Boston Dublin Mumbai Tokyo"
```

cloudera®

Kafka Overview

Chapter 2

cloudera®

Course Chapters

- Introduction
- **Kafka Overview**
- Deploying Apache Kafka
- Kafka Command Line Basics
- Kafka Java API Basics
- Improving Availability through Replication
- Improving Application Reliability
- Improving Application Scalability
- Monitoring and Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka Cluster
- Replicating Kafka to Another Cluster
- Conclusion

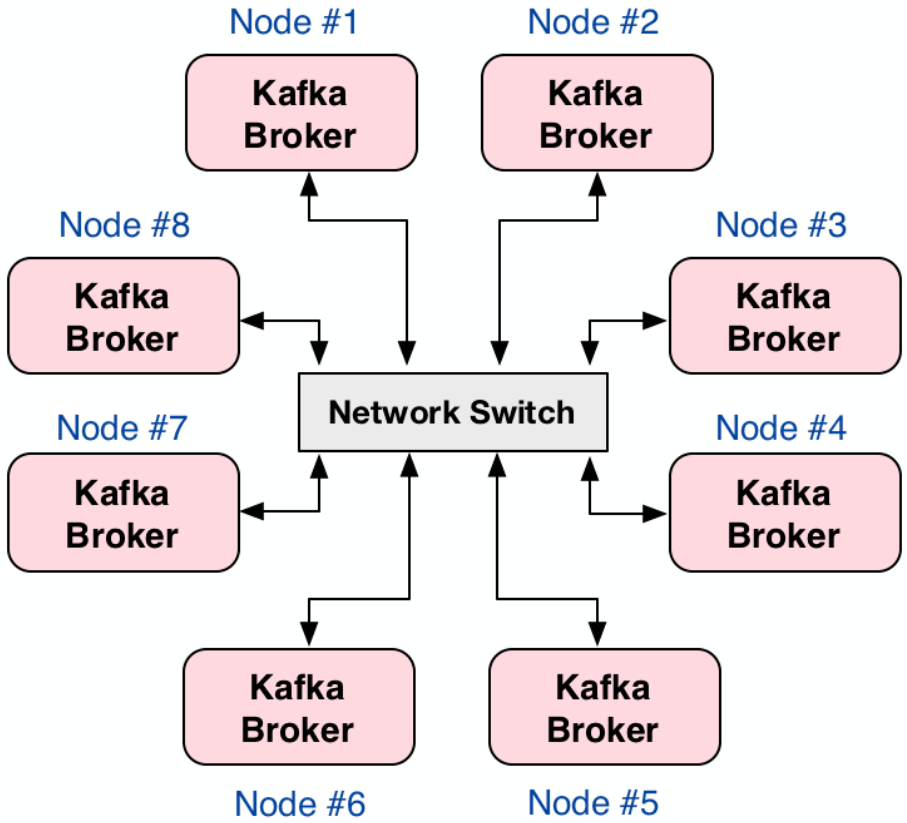
Chapter Topics

Kafka Overview

- **High-Level Architecture**
- Common Use Cases
- Cloudera's Distribution of Apache Kafka
- Essential Points

Kafka Clusters

- **Clusters are composed of interconnected nodes running Kafka software**
  - A *broker* is the Kafka service that listens for client connections

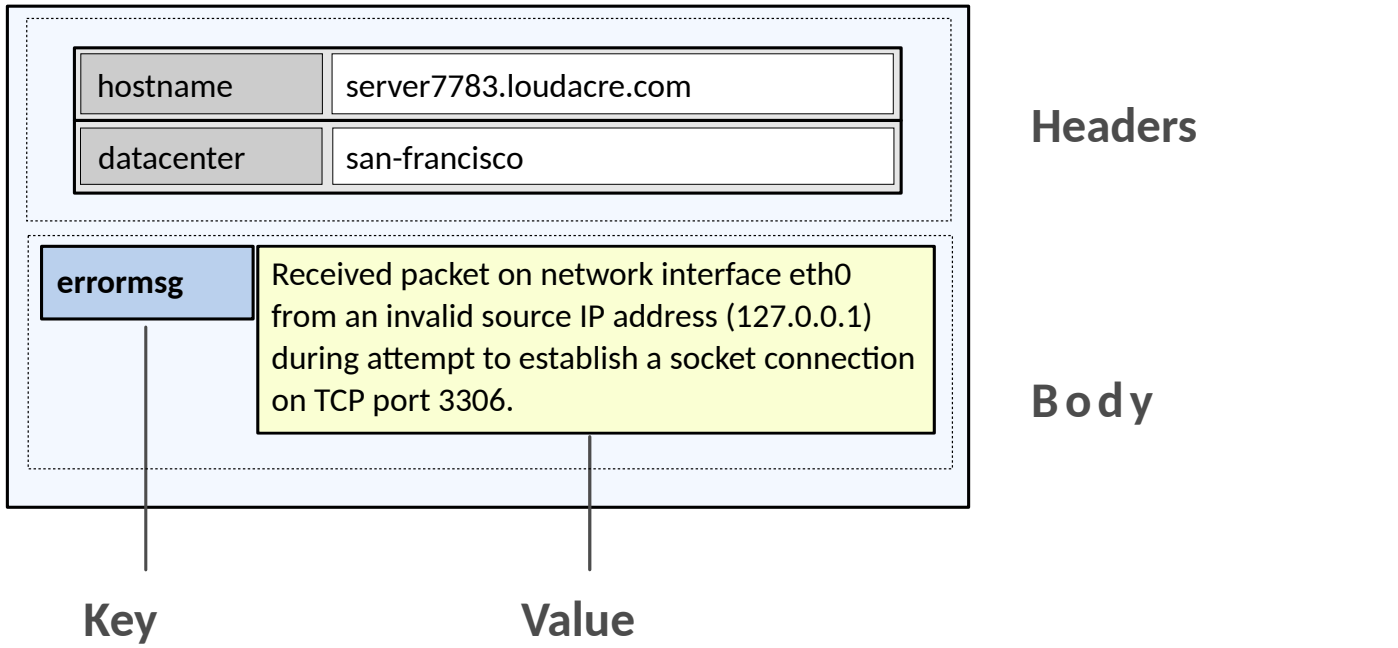


Kafka Messages

- **Messages (also known as records) carry application data**
  - Consist of a key (often empty) and value
  - Typically just a few hundred or thousand bytes
  - Consider 1 MB as a practical maximum size

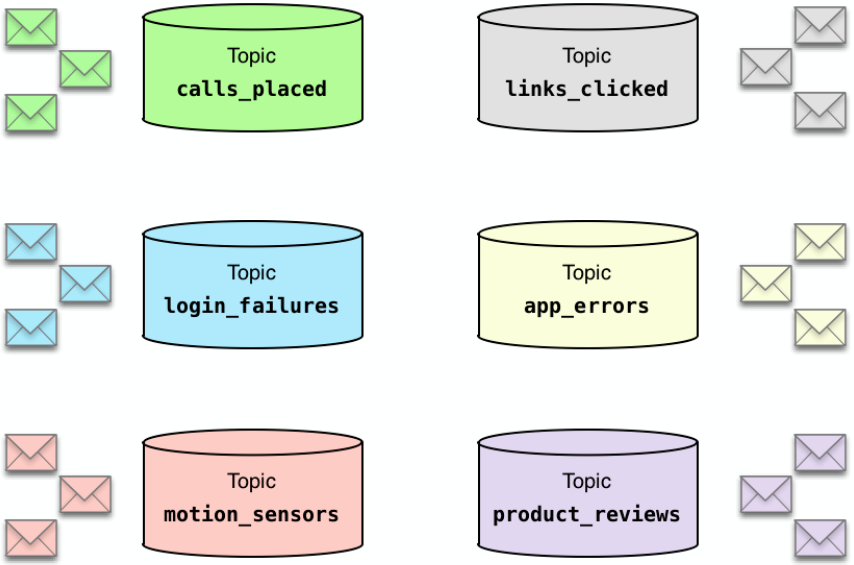
Anatomy of a Kafka Record

- **Headers are an optional set of name-value pairs**
  - Typically used for metadata and message processing
  - Name is of type `String`, value is a `byte[]`
- **The payload is sent in the record's body**
  - Consists of a key (often empty) and value
  - Both can be of any type, but each is ultimately converted to/from `byte[]`



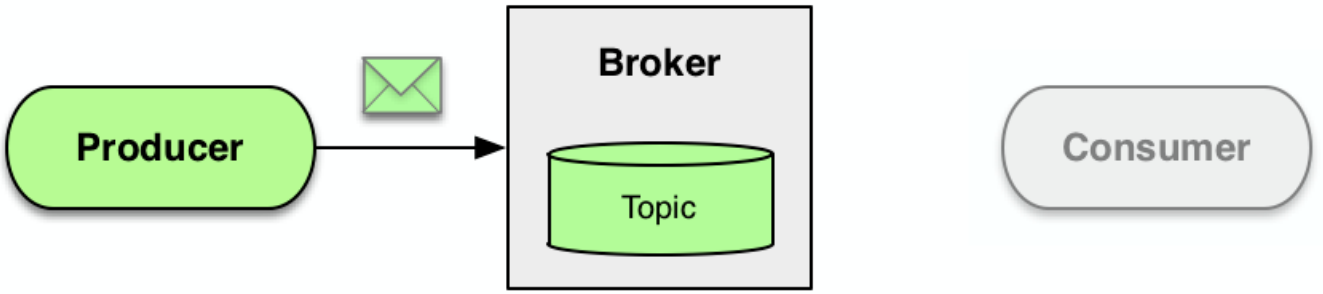
Kafka Topics

- **Messages are classified by categories, known as *topics***
  - Kafka clients specify the topic when sending and receiving messages
  - Kafka brokers use topics to organize and store data



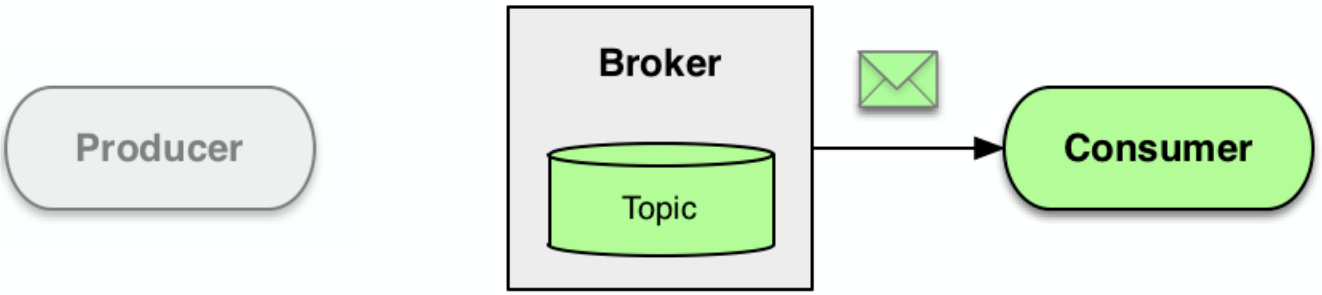
Kafka Producers

- **Producers are Kafka clients that publish messages to a topic**
  - Can be configured to retry if sending fails
  - For efficiency, producers can send a batch of messages to a broker



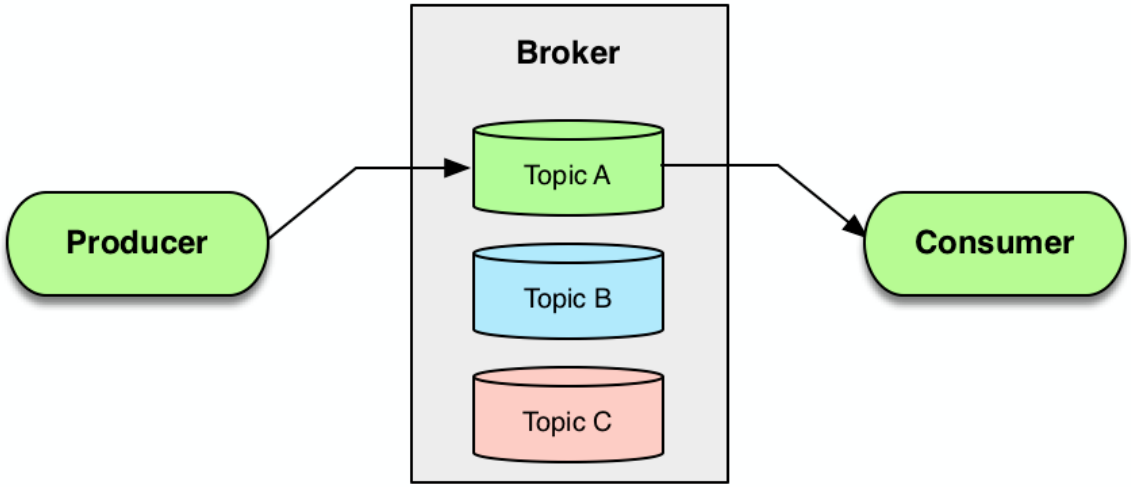
Kafka Consumers

- **Consumers are Kafka clients that fetch messages from a topic**
  - They poll the broker for messages
  - Producers and consumers are decoupled from one another



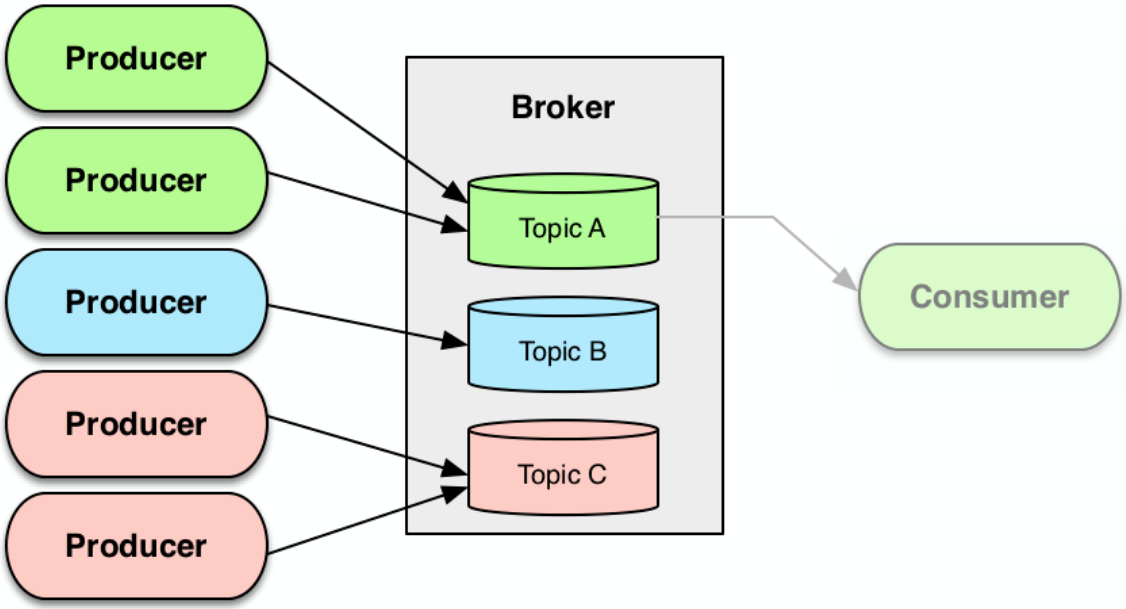
Multiple Topics

- **Each broker may have responsibility for multiple topics**



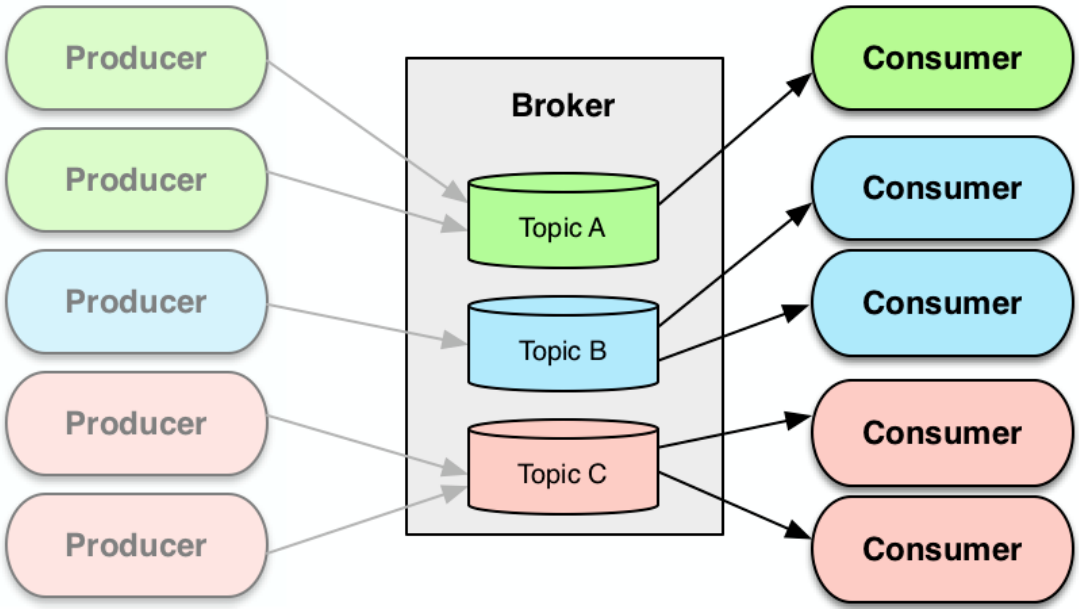
Multiple Producers

- **Each topic may have many producers publishing messages to it**



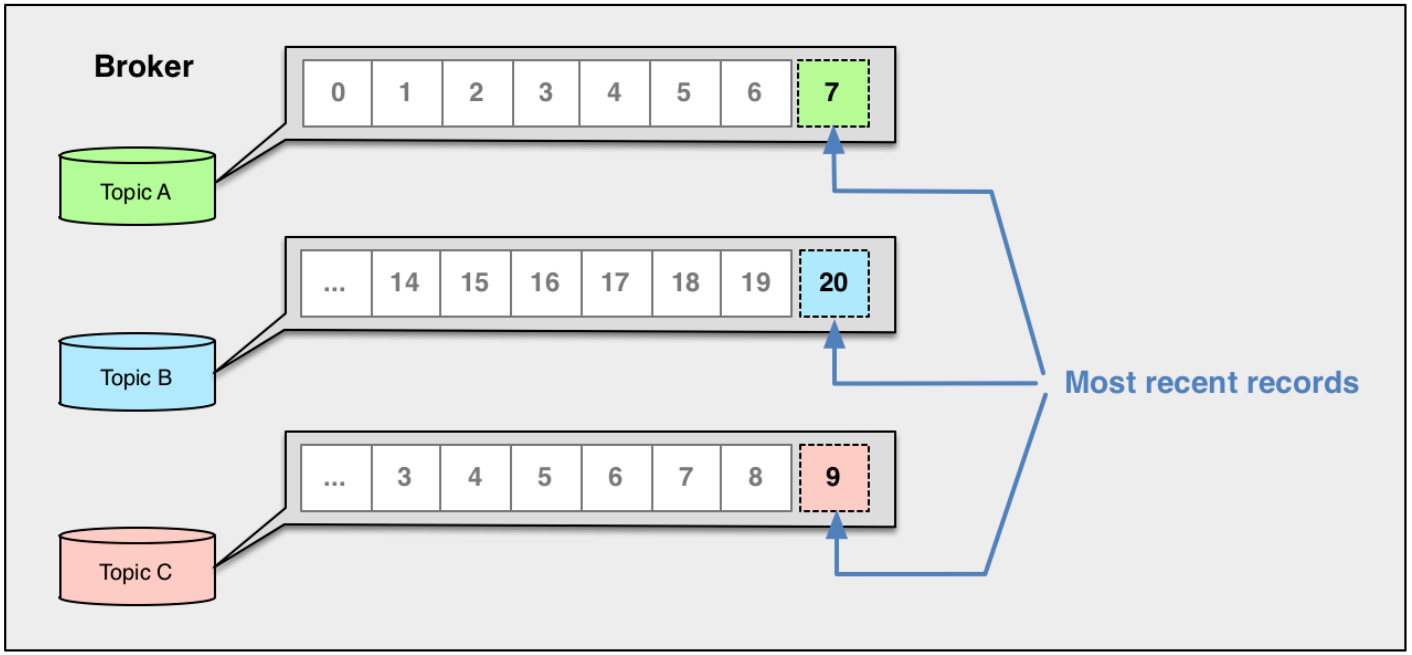
Multiple Consumers

- **Likewise, multiple consumers can read from each topic**
  - Consuming a message does not cause it to be removed



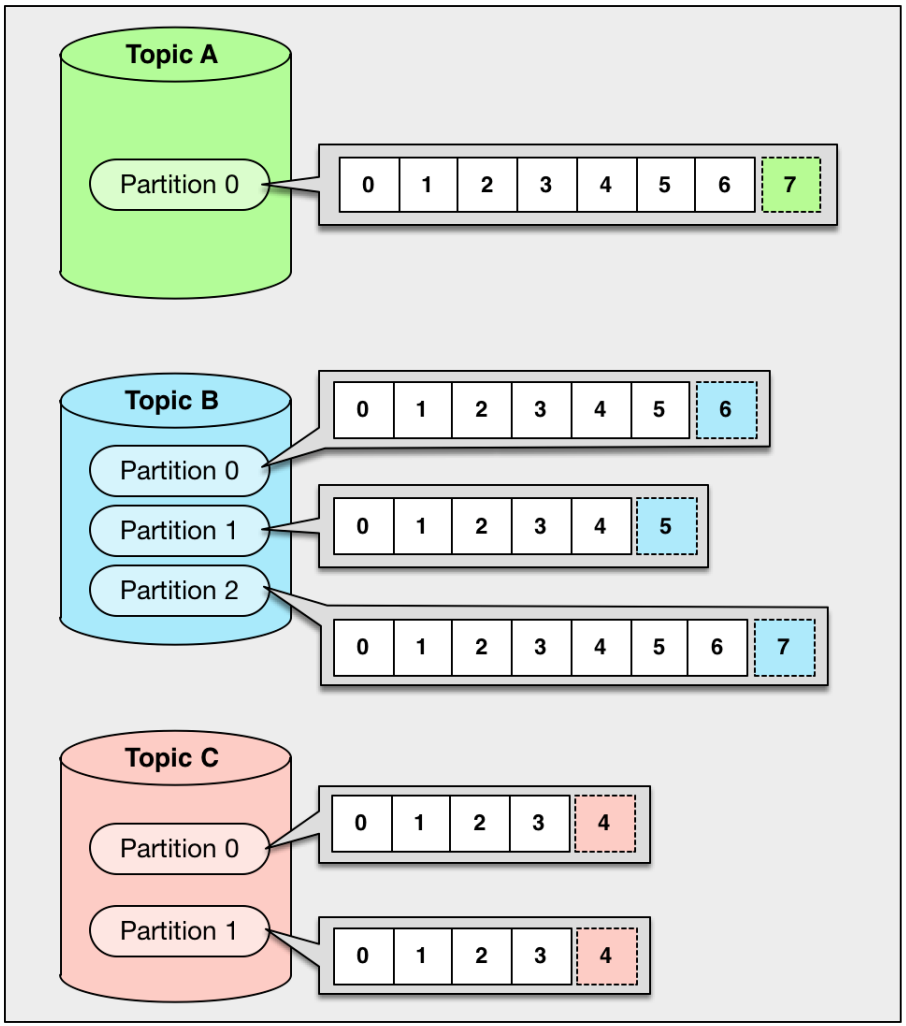
Message Storage and Retention

- **Messages are immutable and stored in the order sent**
  - May be deleted after the specified retention period has elapsed
  - Retention period is configurable on a per-topic basis



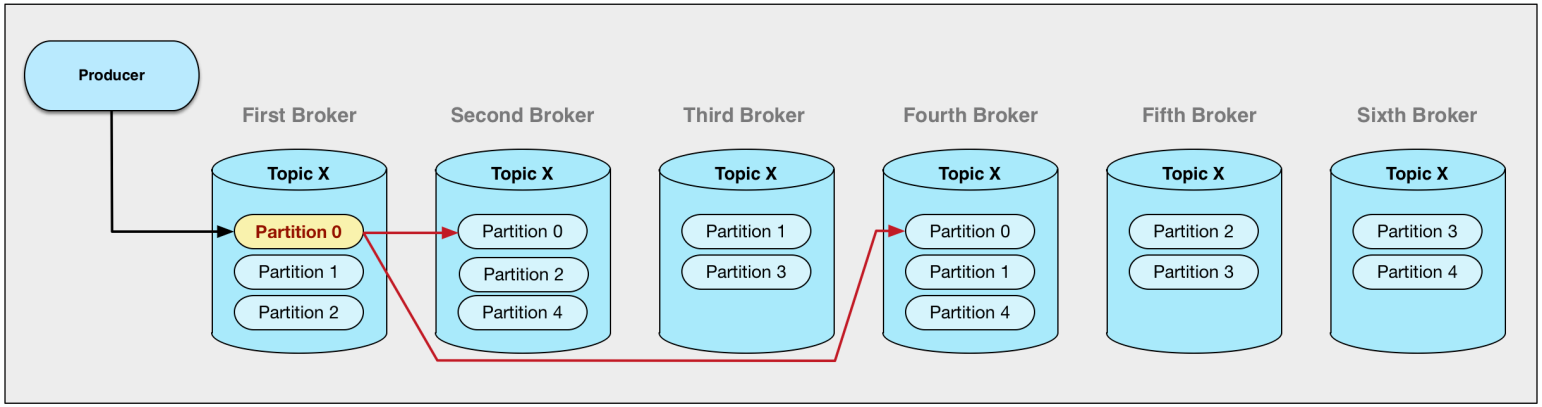
Partitioning

- **For better scalability, each topic can be divided into multiple *partitions***



Replication

- ***Replication* distributes copies of a partition's data across multiple brokers**
  - Provides fault tolerance, based on a topic's specified replication factor
  - This example is a six-node cluster with five partitions and three replicas



Chapter Topics

Kafka Overview

- High-Level Architecture
- **Common Use Cases**
- Cloudera's Distribution of Apache Kafka
- Essential Points

Common Use Cases

- Stream processing
- Data integration
- Messaging
- Log aggregation
- Operational metrics
- Web site activity tracking

Chapter Topics

Kafka Overview

- High-Level Architecture
- Common Use Cases
- **Cloudera's Distribution of Apache Kafka**
- Essential Points

About CDK

- **CDK: Cloudera's Distribution of Apache Kafka**
  - Starting with CDH 6.0.0, it's part of the core parcel
  - Distributed as a separate parcel in CDH 5
- **Each CDK release is derived from an Apache Kafka release**
  - Includes backports and bugfixes from upstream project
  - As with upstream releases, all code is open source
- **Check CDK release notes for important information, which varies by version**
  - Experimental or unsupported features
  - Known issues and workarounds

Understanding CDK Version Numbers

CDK	Version	Release Date
3.1.1	**1.0.1**+kafka3.1.1+3	November 19, 2018
3.1.0	**1.0.1**+kafka3.1.0+35	June 7, 2018
3.0.0	**0.11.0**+kafka3.0.0+50	October 16, 2017
2.2.0	**0.10.2.0**+kafka2.2.0+110	July 13, 2017
2.1.2	**0.10.0.1**+kafka2.1.2+6	October 4, 2017
2.1.1	**0.10.0.0**+kafka2.1.1+21	March 31, 2017
2.1.0	**0.10.0.0**+kafka2.1.0+63	January 31, 2017
2.0.2	**0.9.0.0**+kafka2.0.2+305	July 22, 2016
2.0.1	**0.9.0.0**+kafka2.0.1+283	April 7, 2016
2.0.0	**0.9.0.0**+kafka2.0.0+188	February 19, 2016
1.4.0	**0.8.2.0**+kafka1.4.0+127	December 10, 2015
1.3.2	**0.8.2.0**+kafka1.3.2+116	October 8, 2015
1.3.1	**0.8.2.0**+kafka1.3.1+80	August 3, 2015
1.3.0	**0.8.2.0**+kafka1.3.0+72	April 23, 2015
1.2.0	**0.8.2.0**+kafka1.2.0+57	February 18, 2015

Chapter Topics

Kafka Overview

- High-Level Architecture
- Common Use Cases
- Cloudera's Distribution of Apache Kafka
- **Essential Points**

Essential Points

- **Kafka is a distributed platform for streaming data**
  - Messages (records) are categorized by topic
  - Producers send messages to a topic
  - Consumers read messages from a topic
- **Kafka brokers receive messages from producers**
  - Brokers store those messages and make them available to consumers
  - Topics have a configurable retention period
- **Kafka topics may be divided into partitions to improve scalability**
  - You can protect against data loss by replicating a topic to multiple brokers

Bibliography

The following offer more information on topics discussed in this chapter

- **Apache Kafka Web Site**
  - https://kafka.apache.org/
- **Original LinkedIn Blog Article about Kafka**
  - http://tiny.cloudera.com/kafch02a
- **Cloudera's Distribution of Apache Kafka**
  - http://tiny.cloudera.com/kafch02b
- **Cloudera's Apache Kafka Guide**
  - http://tiny.cloudera.com/kafch02c
- **List of Organizations Using Kafka**
  - http://tiny.cloudera.com/kafch02d

cloudera®

Deploying Apache Kafka

Chapter 3

cloudera®

Course Chapters

- Introduction
- Kafka Overview
- **Deploying Apache Kafka**
- Kafka Command Line Basics
- Kafka Java API Basics
- Improving Availability through Replication
- Improving Application Reliability
- Improving Application Scalability
- Monitoring and Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka Cluster
- Replicating Kafka to Another Cluster
- Conclusion

Chapter Topics

Deploying Apache Kafka

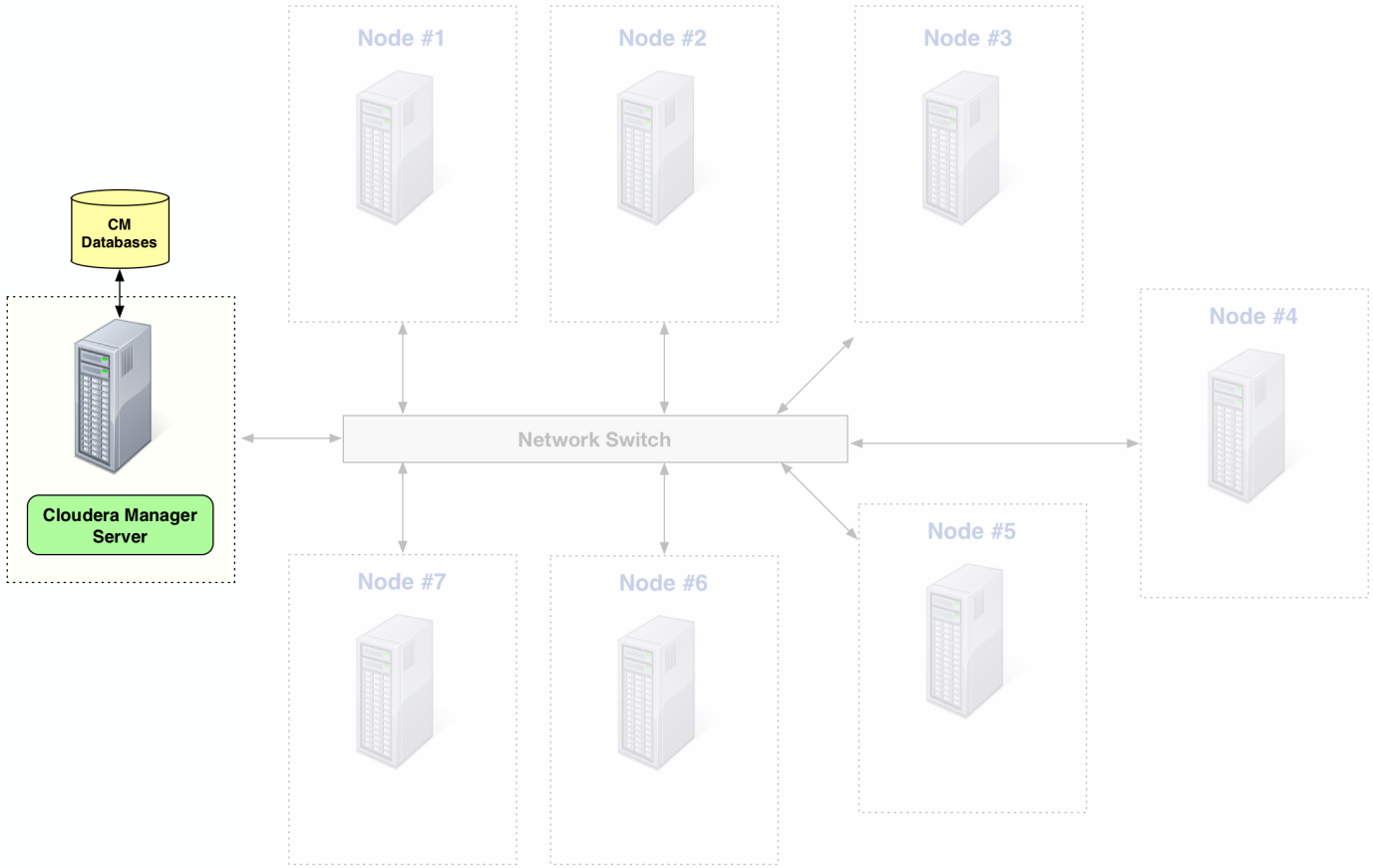
- **System Requirements and Dependencies**
- Service Roles
- Planning Your Deployment
- Deploying Kafka Services
- Hands-On Exercise: Installing the Kafka Service with Cloudera Manager
- Essential Points

CDK System Requirements

- **Requirements vary by version, so be sure to check documentation**
  - Operating system
  - Filesystem
  - JDK

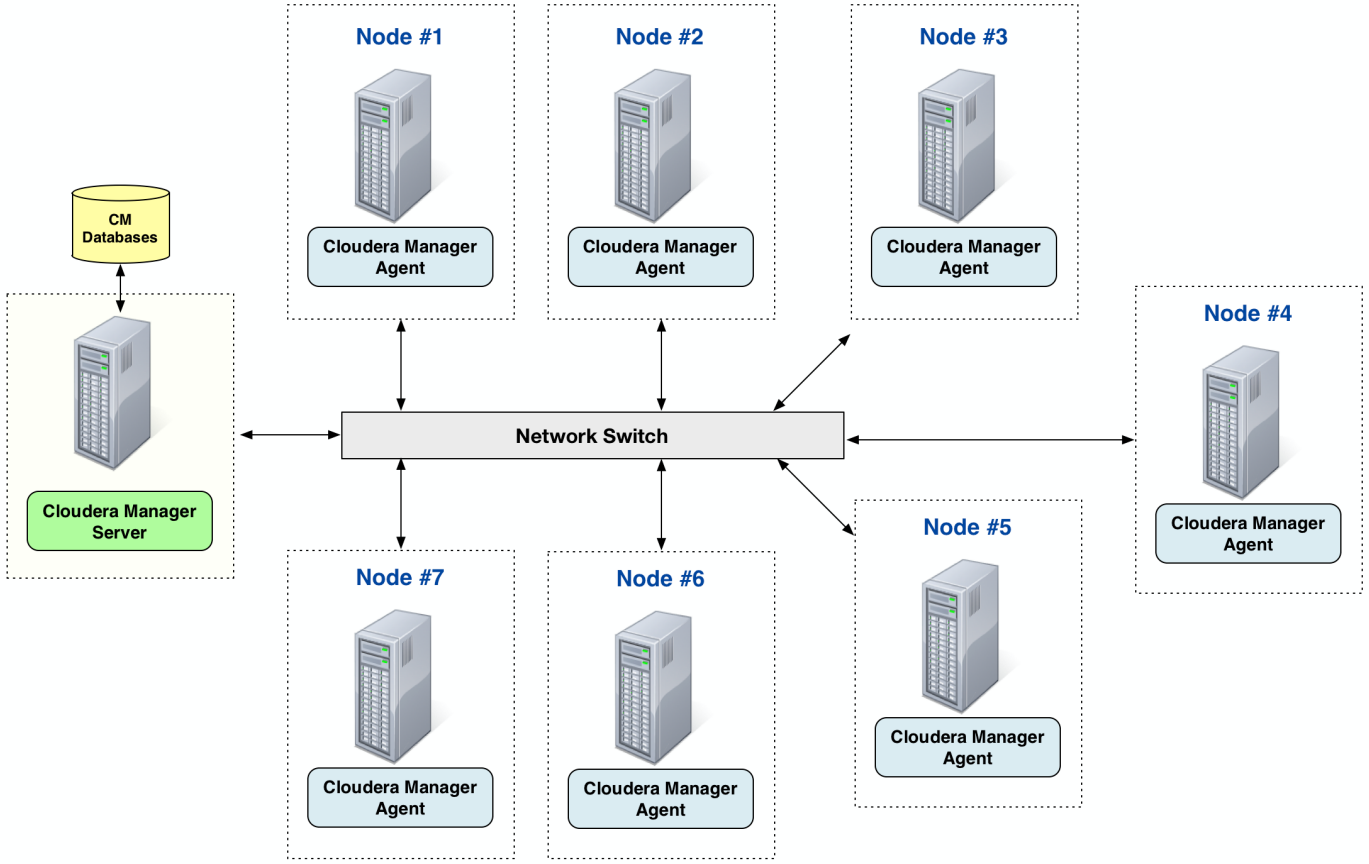
Cloudera Manager: Server

- **Use Cloudera Manager to deploy, monitor, and manage services**
- **Cloudera Manager has a client-server architecture**
  - CM server maintains configurations and provides UI / API for users



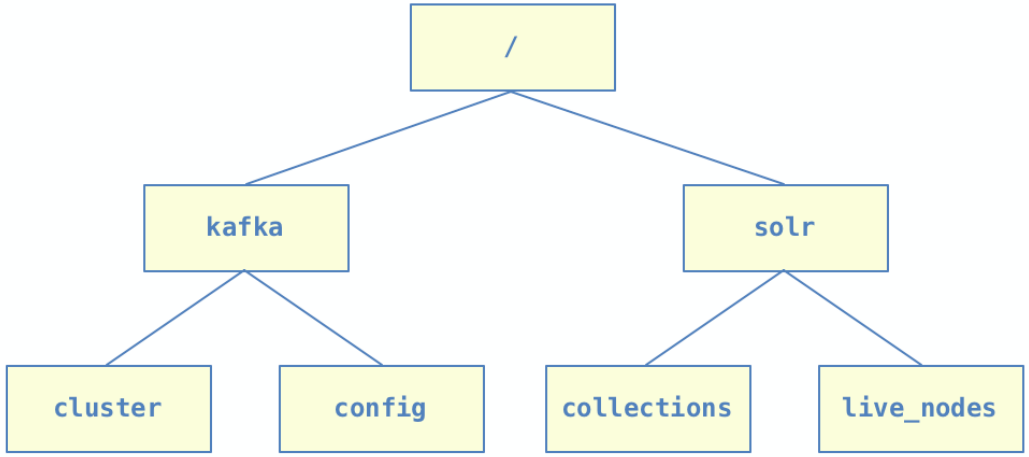
Cloudera Manager: Agents

- **Cloudera Manager agents run on each node in the cluster**
- **Agents collect statistics and manage local services on the node**
  - They receive commands from, and report status to, the server



Service Dependencies: Apache ZooKeeper (Required)

- **Like a few other services, Kafka relies on ZooKeeper**
  - Used to track status
  - Used for coordination within the cluster
  - Used to store metadata
- **ZooKeeper presents a hierarchical data model, composed of *znodes***



Optional Service: Apache Sentry

- **Kafka can also use Apache Sentry for *authorization***
  - Provides support for role-based access control with Kafka
  - There is no technical requirement to install Sentry
  - Security requirements will dictate whether Sentry is needed

Chapter Topics

Deploying Apache Kafka

- System Requirements and Dependencies
- **Service Roles**
- Planning Your Deployment
- Deploying Kafka Services
- Hands-On Exercise: Installing the Kafka Service with Cloudera Manager
- Essential Points

CDK specifies the following three roles

- **Broker**
  - Service that listens for client connections
  - Responsible for data storage and replication
- **Gateway**
  - Client configuration files only (no daemon)
  - Deployed on machines where producers and consumers run
- **MirrorMaker**
  - Service that mirrors topics' content from one cluster to another
  - Destination cluster may be in a remote datacenter
  - Often used to support disaster recovery

Other Service Roles

- **ZooKeeper defines only one role**
  - Server
- **Sentry defines two *roles***
  - Server: Controls access based on defined rules
  - Gateway: Rules are defined using commands run from here

Chapter Topics

Deploying Apache Kafka

- System Requirements and Dependencies
- Service Roles
- **Planning Your Deployment**
- Deploying Kafka Services
- Hands-On Exercise: Installing the Kafka Service with Cloudera Manager
- Essential Points

Available Resources

- **Every cluster has a finite set of resources**
  - Memory
  - Processor
  - Storage
  - Network bandwidth
- **How to best allocate these depends on your specific use case**

Typical Cluster Hardware

- **RAM: 64 GB or more**
- **CPU: Dual processor, eight or more cores each**
- **Storage: 12 or more disks, each at least 1 TB**
  - Do not use network storage (NAS or SAN)
- **Network interface: Gigabit ethernet**
  - 10-gigabit ethernet is becoming more common

Memory Considerations

- **Since Kafka is often constrained by memory, more RAM is usually better**
  - 64 GB should be considered a minimum for new servers
  - With more memory, operating system can do more caching

Processor Considerations

- **Kafka is seldom CPU-bound, so don’t overspend on CPUs**
- **Number of cores matters more than processor speed**
- **Using SSL can significantly increase CPU utilization**

Storage Considerations

- **When choosing the number and size of disks, consider the following**
  - More spindles are better
  - Cloudera does not support drives larger than 8TB
- **Estimate based on message size, send rates, retention, and replication**
  - Plan for future growth
  - Be conservative: More capacity is always better
- **Tradeoffs**
  - Performance (JBOD) versus reliability (RAID 10)
  - Performance of SSDs may not justify their cost
  - SATA drives offer better price/performance than SAS

Planning for ZooKeeper

- **ZooKeeper is designed to be a highly available distributed service**
- **Relies on consensus among the *ensemble* of ZooKeeper servers**
  - This requires a quorum (majority) within the ensemble
  - ZooKeeper instances are therefore deployed in odd numbers
  - Try to spread these across multiple racks or availability zones
- **Three ZooKeeper instances is considered the minimum**
  - Failure of any one of these will not cause downtime
- **Using five ZooKeeper instances is less common**
  - Failure of any two nodes will not cause downtime
  - Ensembles of five nodes may be needed for large clusters
  - Tradeoff: more ZooKeeper instances mean more network traffic

Resources for ZooKeeper

- **ZooKeeper is a fairly lightweight service**
  - Often run on same nodes as Kafka brokers
  - Avoid deploying on machines that have non-uniform or unpredictable loads
- **It is very sensitive to time skew**
  - Large clock corrections will cause problem
  - Strongly recommend using NTP to synchronize all nodes to a common source
- **Disk considerations for ZooKeeper**
  - Does not benefit from SSD
  - Specify a dedicated device (>=1 TB, RAID 0) for *Transaction Log Directory*
  - Set ZooKeeper server's Java heap size large enough to avoid swapping

Chapter Topics

Deploying Apache Kafka

- System Requirements and Dependencies
- Service Roles
- Planning Your Deployment
- **Deploying Kafka Services**
- Hands-On Exercise: Installing the Kafka Service with Cloudera Manager
- Essential Points

Separate Cluster or Shared with CDH?

- **Kafka can coexist alongside other CDH services, such as Spark or Solr**
- **You can also install Kafka in its own cluster**
  - A single Cloudera Manager instance can manage multiple clusters
- **Many customers prefer installing a separate cluster for Kafka**
  - Can restart other services without affecting Kafka (and vice versa)
  - Can optimize hardware selection for Kafka
  - Can optimize settings for Kafka
  - These benefits come with a small amount of overhead and complexity

Chapter Topics

Deploying Apache Kafka

- System Requirements and Dependencies
- Service Roles
- Planning Your Deployment
- Deploying Kafka Services
- **Hands-On Exercise: Installing the Kafka Service with Cloudera Manager**
- Essential Points

Hands-On Exercise: Installing the Kafka Service with Cloudera Manager

- **In this exercise, you will create a new cluster, and then install Kafka and ZooKeeper services on that cluster**
- **Exercise directory: `exercise-code/installing-kafka`**

Chapter Topics

Deploying Apache Kafka

- System Requirements and Dependencies
- Service Roles
- Planning Your Deployment
- Deploying Kafka Services
- Hands-On Exercise: Installing the Kafka Service with Cloudera Manager
- **Essential Points**

Essential Points

- **Kafka relies on Apache ZooKeeper for metadata and coordination**
  - ZooKeeper ensembles should always have an odd number of nodes
  - Three ZooKeeper nodes is sufficient for most clusters
- **For maximum performance and availability, deploy Kafka on its own cluster**
  - It will remain unaffected by restarts, upgrades, and load related to other CDH services

Bibliography

The following offer more information on topics discussed in this chapter

- **Cloudera Documentation: CDK Requirements and Supported Versions**
  - http://tiny.cloudera.com/kafch03a
- **Cloudera Documentation: Installing, Migrating and Upgrading Kafka**
  - http://tiny.cloudera.com/kafch03b
- **FAQ: Recommendations on Optimizing Hardware for Reliability**
  - http://tiny.cloudera.com/kafch03c

cloudera®

Kafka Command Line Basics

Chapter 4

cloudera®

Course Chapters

- Introduction
- Kafka Overview
- Deploying Apache Kafka
- **Kafka Command Line Basics**
- Kafka Java API Basics
- Improving Availability through Replication
- Improving Application Reliability
- Improving Application Scalability
- Monitoring and Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka Cluster
- Replicating Kafka to Another Cluster
- Conclusion

cloudera

Chapter Topics

Kafka Command Line Basics

- **Kafka Topic Overview and Best Practices**
- Topic Management
- Hands-On Exercise: Managing Topics using the CLI
- Using Producers and Consumers
- Hands-On Exercise: Connecting Producers and Consumers from the Command Line
- Message Retention
- Essential Points

Topics: Recap

- **Topics are used to categorize messages**
  - Divided into partitions
  - May be replicated to other nodes
- **By default, Kafka allows topic auto-creation**
  - We recommend disabling this feature

Considerations for Topic Creation

- **Partition count and replication factor are set when topic is created**
  - Both can have a major impact on performance
- **Topic names may contain letters, numbers, underscores, dots, and hyphens**
  - Names are case-sensitive
  - Simple names are often similar to database/table naming conventions
  - More advanced cases may have dot-separated hierarchical names
- **No hard limits, but scaling beyond a few thousand topics is hard**

Chapter Topics

Kafka Command Line Basics

- Kafka Topic Overview and Best Practices
- **Topic Management**
- Hands-On Exercise: Managing Topics using the CLI
- Using Producers and Consumers
- Hands-On Exercise: Connecting Producers and Consumers from the Command Line
- Message Retention
- Essential Points

Aside: Specifying Hostnames in Command-Line Utilities

- Kafka's command-line utilities require access to Kafka or ZooKeeper
- Typing these each time is tedious and error prone
- We recommend using environment variables to set these
- Note the syntax of ZooKeeper connection string (znode after last host only)

Chapter Topics

Kafka Command Line Basics

- Kafka Topic Overview and Best Practices
- Topic Management
- **Hands-On Exercise: Managing Topics using the CLI**
- Using Producers and Consumers
- Hands-On Exercise: Connecting Producers and Consumers from the Command Line
- Message Retention
- Essential Points

Hands-On Exercise: Managing Topics using the CLI

- **In this exercise, you will create, list, describe, and delete Kafka topics by using a command-line utility**
- **Exercise directory: `exercise-code/managing-topics-cli`**

Chapter Topics

Kafka Command Line Basics

- Kafka Topic Overview and Best Practices
- Topic Management
- Hands-On Exercise: Managing Topics using the CLI
- **Using Producers and Consumers**
- Hands-On Exercise: Connecting Producers and Consumers from the Command Line
- Message Retention
- Essential Points

Chapter Topics

Kafka Command Line Basics

- Kafka Topic Overview and Best Practices
- Topic Management
- Hands-On Exercise: Managing Topics using the CLI
- Using Producers and Consumers
- **Hands-On Exercise: Connecting Producers and Consumers from the Command Line**
- Message Retention
- Essential Points

Hands-On Exercise: Connecting Producers and Consumers from the Command Line

- **In this exercise, you will use the command-line producer utility to interactively send messages to a Kafka topic and the command-line consumer utility to read those messages from the topic**
- **Exercise directory: `exercise-code/producer-consumer-cli`**

Chapter Topics

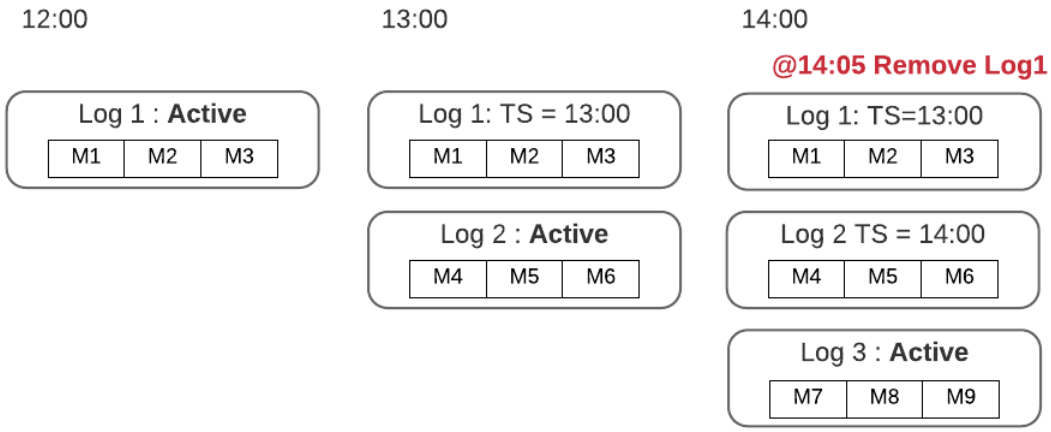
Kafka Command Line Basics

- Kafka Topic Overview and Best Practices
- Topic Management
- Hands-On Exercise: Managing Topics using the CLI
- Using Producers and Consumers
- Hands-On Exercise: Connecting Producers and Consumers from the Command Line
- **Message Retention**
- Essential Points

Message Retention

- **Messages in Kafka are not stored indefinitely**
  - Duration and size of storage is *influenced* by `retention.ms` and `retention.bytes` properties
- **Longer retention periods require more disk space**
- **Shorter retention periods can mean data loss if messages expire before consumption**
- **Important: Message retention applies on a per-partition basis**
  - A topic having 10 partitions and `retention.bytes` of 1 GB could require at least 10 GB

Message Retention



Demo: Message Retention Period

- **Demonstrating the Effect of the Message Retention Period**
- **Demo directory: `demos/retention-period`**

Chapter Topics

Kafka Command Line Basics

- Kafka Topic Overview and Best Practices
- Topic Management
- Hands-On Exercise: Managing Topics using the CLI
- Using Producers and Consumers
- Hands-On Exercise: Connecting Producers and Consumers from the Command Line
- Message Retention
- **Essential Points**

Bibliography

The following offer more information on topics discussed in this chapter

- **Kafka Topic Naming Conventions**
  - http://tiny.cloudera.com/kafch04a
- **Using Apache Kafka Command-line Tools**
  - http://tiny.cloudera.com/kafch04b

cloudera®

Kafka Java API Basics

Chapter 5

cloudera®

Course Chapters

- Introduction
- Kafka Overview
- Deploying Apache Kafka
- Kafka Command Line Basics
- **Kafka Java API Basics**
- Improving Availability through Replication
- Improving Application Reliability
- Improving Application Scalability
- Monitoring and Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka Cluster
- Replicating Kafka to Another Cluster
- Conclusion

Chapter Topics

Kafka Java API Basics

- **Overview of Kafka's APIs**
- Topic Management from the Java API
- Hands-On Exercise: Managing Kafka Topics Using the Java API
- Using Producers and Consumers from the Java API
- Hands-On Exercise: Developing Producers and Consumers with the Java API
- Hands-On Exercise: Basic Producer Options with the Java API
- Essential Points

History of Kafka Client APIs

- **Completely rewritten prior to 1.0 release**
- **Originally written in Scala**
  - Package names start with `kafka`
- **These "old APIs" are now deprecated**
  - Do not use for new code
  - Migrate existing code to "new APIs" as soon as possible
- **The new APIs are much easier to use**

API Documentation

- **Kafka's API documentation is informative**
  - Most package-level index pages have brief descriptions of classes
  - Most important classes and interfaces have additional explanations
- **There are three types of clients, which correspond to these three packages**
  - `org.apache.kafka.clients.admin`
  - `org.apache.kafka.clients.consumer`
  - `org.apache.kafka.clients.producer`
- **You'll also use classes in `org.apache.kafka.common` package**
  - Contains exceptions, metrics, and other classes used by client code

Chapter Topics

Kafka Java API Basics

- Overview of Kafka's APIs
- **Topic Management from the Java API**
- Hands-On Exercise: Managing Kafka Topics Using the Java API
- Using Producers and Consumers from the Java API
- Hands-On Exercise: Developing Producers and Consumers with the Java API
- Hands-On Exercise: Basic Producer Options with the Java API
- Essential Points

Admin API

- **Topic management is done by using the Admin API**
  - Defined in the `org.apache.kafka.clients.admin` package
- **Relatively new API, which is still undergoing changes**
- **Enables Java code to create, list, and describe topics**
- **Communicates directly with brokers, not ZooKeeper**

Chapter Topics

Kafka Java API Basics

- Overview of Kafka's APIs
- Topic Management from the Java API
- **Hands-On Exercise: Managing Kafka Topics Using the Java API**
- Using Producers and Consumers from the Java API
- Hands-On Exercise: Developing Producers and Consumers with the Java API
- Hands-On Exercise: Basic Producer Options with the Java API
- Essential Points

Hands-On Exercise: Managing Kafka Topics Using the Java API

- **In this exercise, you will use the Java API to create, list, and delete Kafka topics**
- **Exercise directory: `exercise-code/managing-topics-java`**

Chapter Topics

Kafka Java API Basics

- Overview of Kafka's APIs
- Topic Management from the Java API
- Hands-On Exercise: Managing Kafka Topics Using the Java API
- **Using Producers and Consumers from the Java API**
- Hands-On Exercise: Developing Producers and Consumers with the Java API
- Hands-On Exercise: Basic Producer Options with the Java API
- Essential Points

Consumer API

- **Introduced in Kafka 0.9**
- **Package is `org.apache.kafka.clients.consumer`**
- **Three essential classes**
  - `KafkaConsumer<K,V>`
  - `ConsumerConfig`
  - `ConsumerRecord`

Producer API

- **Introduced in Kafka 0.8**
- **Package is `org.apache.kafka.clients.producer`**
- **Three essential classes**
  - `KafkaProducer<K,V>`
  - `ProducerConfig`
  - `ProducerRecord`

Three Styles of Producer

- **Fire and Forget**
- **Synchronous**
- **Asynchronous**
  - Uses an interface (`Callback` defined in Producer API)

Chapter Topics

Kafka Java API Basics

- Overview of Kafka's APIs
- Topic Management from the Java API
- Hands-On Exercise: Managing Kafka Topics Using the Java API
- Using Producers and Consumers from the Java API
- **Hands-On Exercise: Developing Producers and Consumers with the Java API**
- Hands-On Exercise: Basic Producer Options with the Java API
- Essential Points

Hands-On Exercise: Developing Producers and Consumers with the Java API

- **In this exercise, you will work with Java code that implements simple Kafka consumer and producers with the Java API**
- **Exercise directory: `exercise-code/producer-consumer-java`**

Chapter Topics

Kafka Java API Basics

- Overview of Kafka's APIs
- Topic Management from the Java API
- Hands-On Exercise: Managing Kafka Topics Using the Java API
- Using Producers and Consumers from the Java API
- Hands-On Exercise: Developing Producers and Consumers with the Java API
- **Hands-On Exercise: Basic Producer Options with the Java API**
- Essential Points

Hands-On Exercise: Basic Producer Options with the Java API

- **In this exercise, you will work with Java code for three different types of Kafka producers**
- **Exercise directory: `exercise-code/basic-producer-options`**

Chapter Topics

Kafka Java API Basics

- Overview of Kafka's APIs
- Topic Management from the Java API
- Hands-On Exercise: Managing Kafka Topics Using the Java API
- Using Producers and Consumers from the Java API
- Hands-On Exercise: Developing Producers and Consumers with the Java API
- Hands-On Exercise: Basic Producer Options with the Java API
- **Essential Points**

Essential Points

- **As a developer, you will use Kafka's client APIs**
- **These APIs were changed prior to 1.0**
  - New API package names begin with `org.apache.kafka`
  - Old API is deprecated
  - Always use the *new* API
- **There are three client APIs**
  - Admin
  - Consumer
  - Producer

Bibliography

The following offer more information on topics discussed in this chapter

- **API Section in Apache Kafka 1.0 Documentation**
  - http://tiny.cloudera.com/kafch05a
- **Apache Kafka 1.0 Java API Documentation**
  - http://tiny.cloudera.com/kafch05b
- **Blog post about New Consumer API (Introduced in Kafka 0.9)**
  - http://tiny.cloudera.com/kafch05c

cloudera®

Improving Availability through Replication

Chapter 6

cloudera®

Course Chapters

- Introduction
- Kafka Overview
- Deploying Apache Kafka
- Kafka Command Line Basics
- Kafka Java API Basics
- **Improving Availability through Replication**
- Improving Application Reliability
- Improving Application Scalability
- Monitoring and Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka Cluster
- Replicating Kafka to Another Cluster
- Conclusion

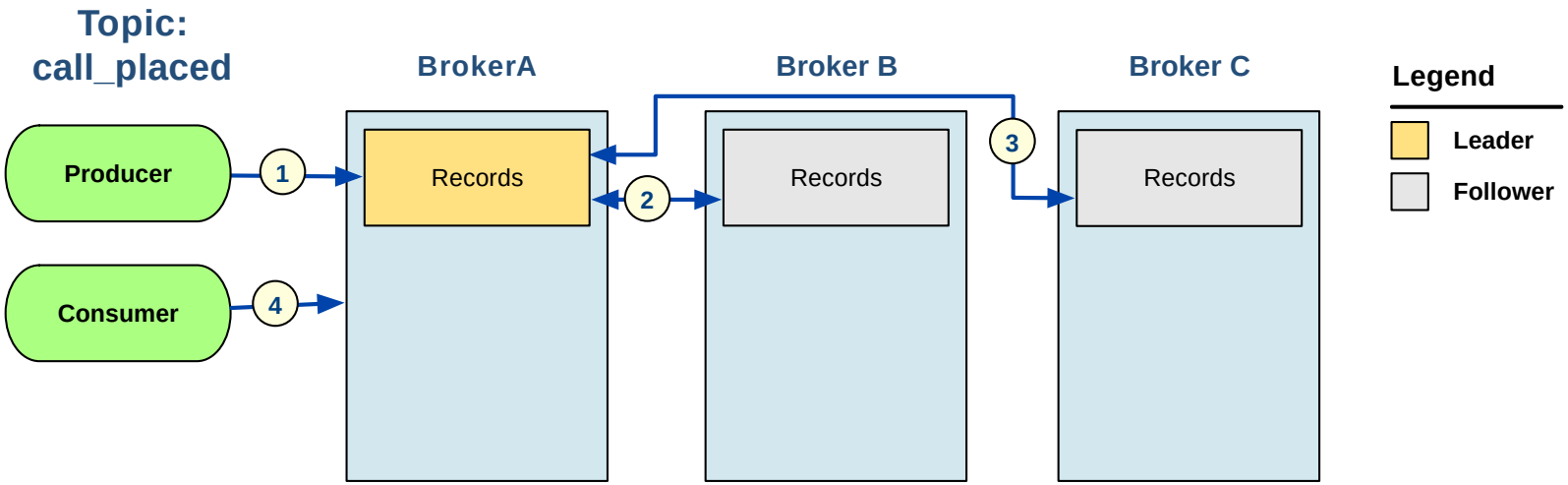
Chapter Topics

Improving Availability through Replication

- **Replication Overview**
- Hands-On Exercise: Observing Downtime Due to Broker Failure
- Considerations for the Replication Factor
- Hands-On Exercise: Adding Replicas to Improve Availability
- Essential Points

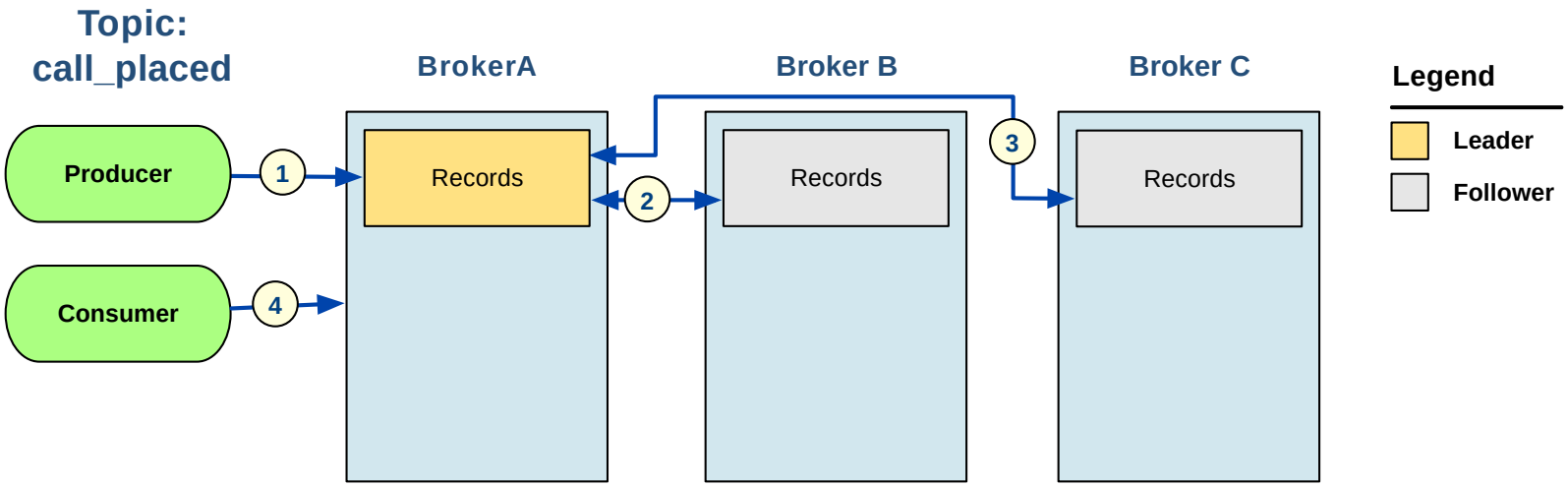
Replication

- **All Kafka topics are assigned a *replication factor***
  - Additional disk space and network traffic is traded for fault tolerance
  - Brokers are assigned to be a leader or a follower for each replica
  - Replication factor can be assigned per topic
  - Consumers and producers write and read from only the leader



Replication (Cont'd)

- **Followers periodically fetch data from the leader**
  - Followers send "fetch requests" to the leader
  - Leaders send the followers new messages received since the last fetch



Fault Tolerance

- **All brokers must heartbeat to ZooKeeper**
  - If a leader does not heartbeat to ZooKeeper within a specified time, it is regarded as dead
  - Kafka controller elects one of the followers as the new leader
  - Producers and consumers periodically fetch metadata and will recognize this election
  - Producers and consumers will begin sending and receiving messages from the new leader

Fault Tolerance (Cont'd)

- **Restoring Replication Count**
  - If the previous leader for a replica becomes in-sync, it is elected again as the leader (known as "Preferred Leader Election") `auto.leader.rebalance.enable`
  - Kafka currently does not support automatic *assignment* of new followers *
  - Administrators can assign another broker to be a replica of a topic using the `kafka-reassign-partitions` command ‡
  - Alternatively, a new broker can be brought up, and assigned the ID of the previous broker †
    - New broker will inherit the same leader/follower assignments as previous broker, and begin replicating data

*https://cwiki.apache.org/confluence/display/KAFKA/KIP-46%3A+Self+Healing+Kafka
†https://www.cloudera.com/documentation/enterprise/6/6.1/topics/kafka_admin_migration.html#kafka_admin
‡https://www.cloudera.com/documentation/enterprise/6/6.0/topics/kafka_admin.html#kafka_cli__kafka_reassign_partitions_command

Chapter Topics

Improving Availability through Replication

- Replication Overview
- **Hands-On Exercise: Observing Downtime Due to Broker Failure**
- Considerations for the Replication Factor
- Hands-On Exercise: Adding Replicas to Improve Availability
- Essential Points

Hands-On Exercise: Observing Downtime Due to Broker Failure

- **In this exercise, you will see the effect of Kafka broker failure when topics are configured with only one replica**
- **Exercise directory: `exercise-code/broker-failure`**

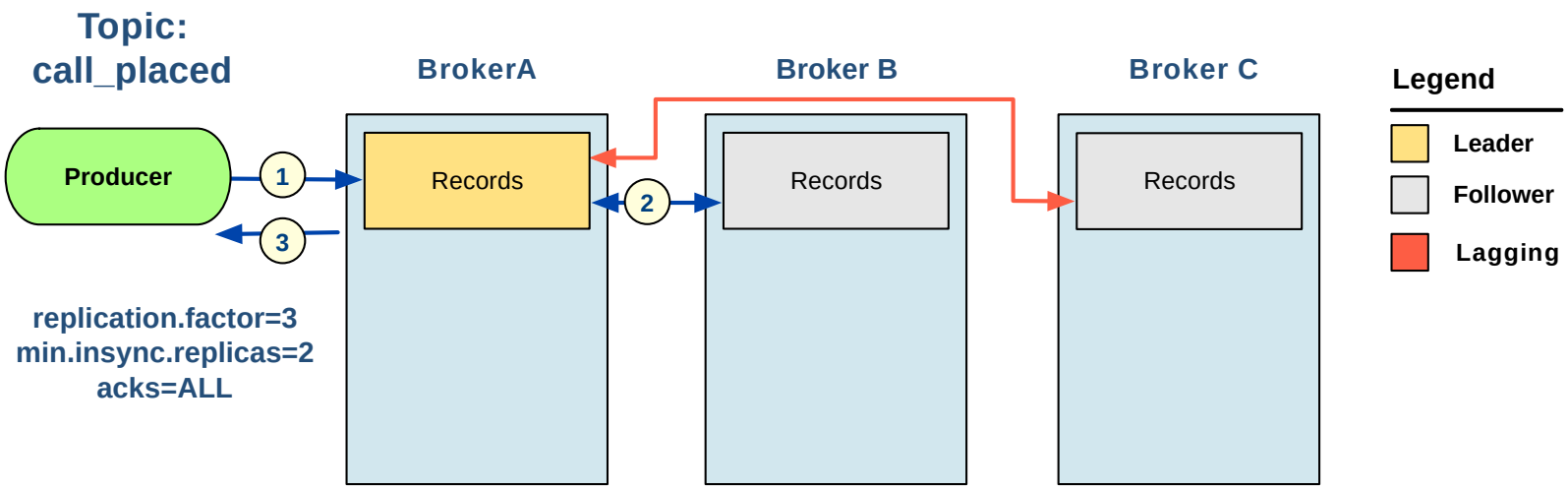
Chapter Topics

Improving Availability through Replication

- Replication Overview
- Hands-On Exercise: Observing Downtime Due to Broker Failure
- **Considerations for the Replication Factor**
- Hands-On Exercise: Adding Replicas to Improve Availability
- Essential Points

Replication Factor Considerations

- A replication factor of at least 3 is recommended. However, this increases latency for producers opting to wait for acknowledgement for delivery to all replicas
- An optional setting `min.insync.replicas` is available as a tradeoff
- Set `min.insync.replicas` to a high enough value to provide data safety (typically 2 with replication.factor=3)
- Producers using `acks=all` will receive acknowledgement from the leader when acknowledgement is received from `min.insync.replicas`



Chapter Topics

Improving Availability through Replication

- Replication Overview
- Hands-On Exercise: Observing Downtime Due to Broker Failure
- Considerations for the Replication Factor
- **Hands-On Exercise: Adding Replicas to Improve Availability**
- Essential Points

Hands-On Exercise: Adding Replicas to Improve Availability

- **In this exercise, you will create a topic whose data is replicated to multiple nodes, and then verify that killing a Kafka broker does not result in downtime**
- **Exercise directory: `exercise-code/adding-replicas`**

Chapter Topics

Improving Availability through Replication

- Replication Overview
- Hands-On Exercise: Observing Downtime Due to Broker Failure
- Considerations for the Replication Factor
- Hands-On Exercise: Adding Replicas to Improve Availability
- **Essential Points**

Essential Points

- **Replication is simply the copying of messages to multiple brokers**
  - Protects against permanent data loss
  - Ensures availability of data despite server failure
  - Replication factor is a tradeoff between performance and data reliability
  - Replication factor is set on a per-topic basis
- **Replication is performed by Kafka brokers**
  - Producers and consumers do not participate in replication
  - Replication is implemented with a leader-follower pattern
  - Followers fetch data from leaders; leaders do not push data to followers
- **Kafka assigns leader/follower replicas at topic creation time**
  - Brokers assigned as replicas are basically "static"
  - Kafka controller performs leader reassignment if a leader lags or becomes unavailable
  - Manual intervention is required if a broker is permanently down

Bibliography

The following offer more information on topics discussed in this chapter

- **Entertaining "Balancing Apache Kafka Clusters" Video**
  - `Balancing Apache Kafka Clusters`

cloudera®

Improving Application Reliability

Chapter 7

cloudera®

Course Chapters

- Introduction
- Kafka Overview
- Deploying Apache Kafka
- Kafka Command Line Basics
- Kafka Java API Basics
- Improving Availability through Replication
- **Improving Application Reliability**
- Improving Application Scalability
- Monitoring and Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka Cluster
- Replicating Kafka to Another Cluster
- Conclusion

Chapter Topics

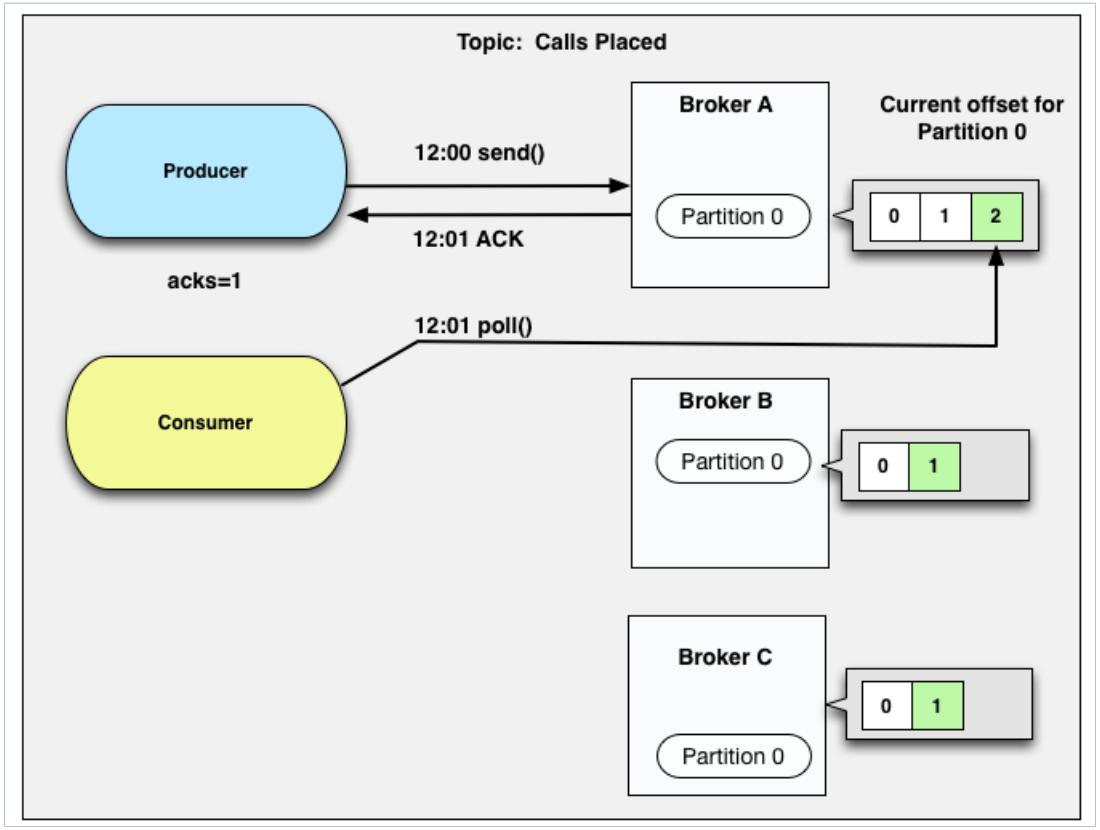
Improving Application Reliability

- **Commits**
- Batching
- Transactions
- Handling Consumer Failure
- Offset Management
- Hands-On Exercise: Detecting and Suppressing Duplicate Messages
- Hands-On Exercise: Handling Invalid Records
- Handling Producer Failure
- Essential Points

Producers are Responsible For Ensuring Message Delivery

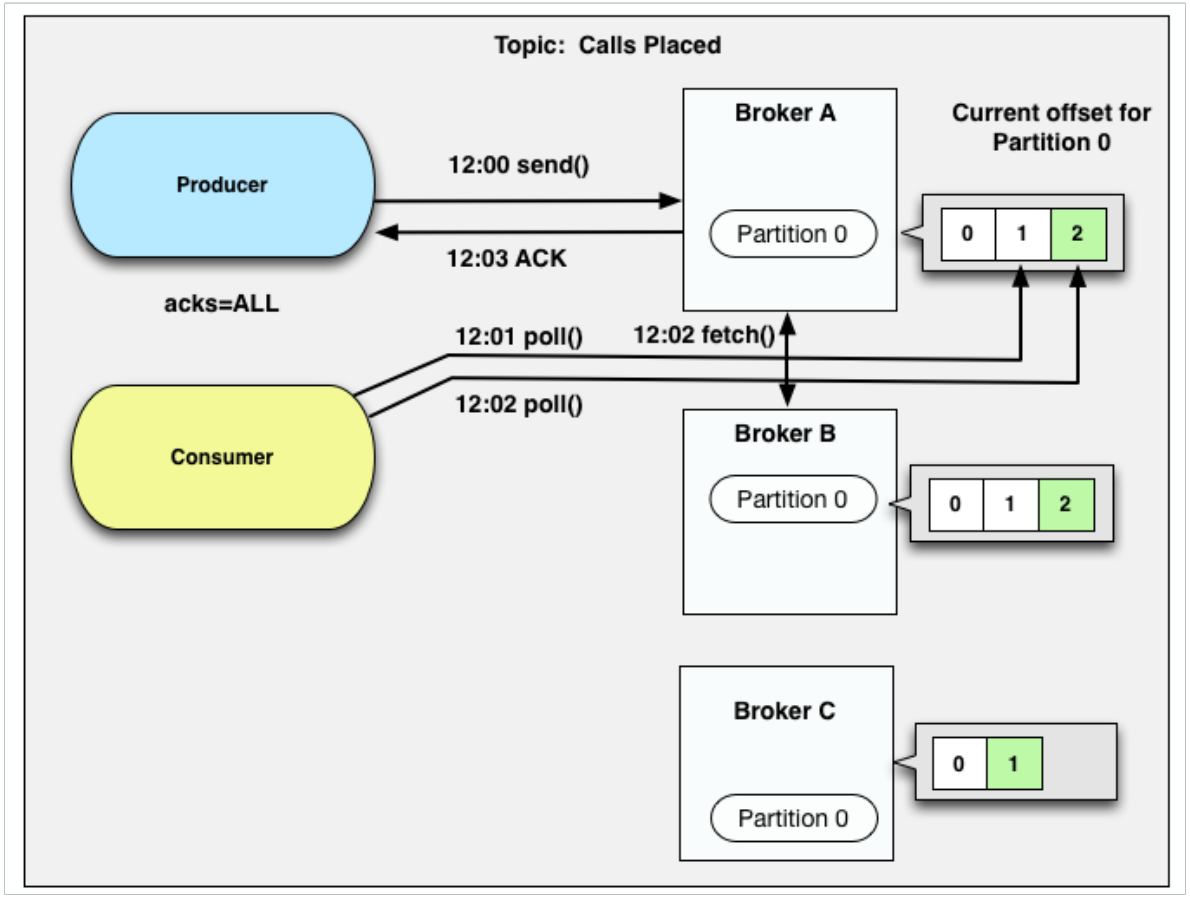
- **Producers using `acks=0` (Fire and Forget) send the message and immediately continue**
  - No acknowledgement is given
  - Messages may be lost for many reasons (broker failure, network timeout, etc)
  - Provides very fast delivery, but no guarantee of delivery
- **Producers using `acks=1` will receive acknowledgement from the leader**
  - Only one replica is acknowledged. Not Recommended!
  - Leader will regard the message as committed
  - Consumers can view committed messages on the leader
  - However, if leader fails, no guarantee that message exists on other brokers

Example of acks=1



Producers are Responsible For Ensuring Message Delivery

- **Producers should use ACKS="all"**



Recommended Producer Settings for Commits

- **`acks/ProducerConfig.ACKS_CONFIG`: Producers should use ACKS="all"**
  - Producers will wait for `min.insync.replicas` for a topic to confirm message delivery
- **`retries/ProducerConfig.RETRIES_CONFIG`: Number of times to retry a failed send()**
  - Recommendation: `Long.MAX_VALUE`
- **`max.in.flight.requests.per.connection/ MAX_IN_FLIGHT_REQUESTS_PER_CONNECTION`:**
  - Maximum number of unaknowledged requests the producer will send on a single connection
  - Protects against out-of-order messages from being sent
  - Recommendation: 1

Demo: ISRs vs. Acks

- In this demo, you will see the difference between producers using ACKS=1 versus ACKS=all
- Demo directory: `demos/isrs-vs-acks`

Chapter Topics

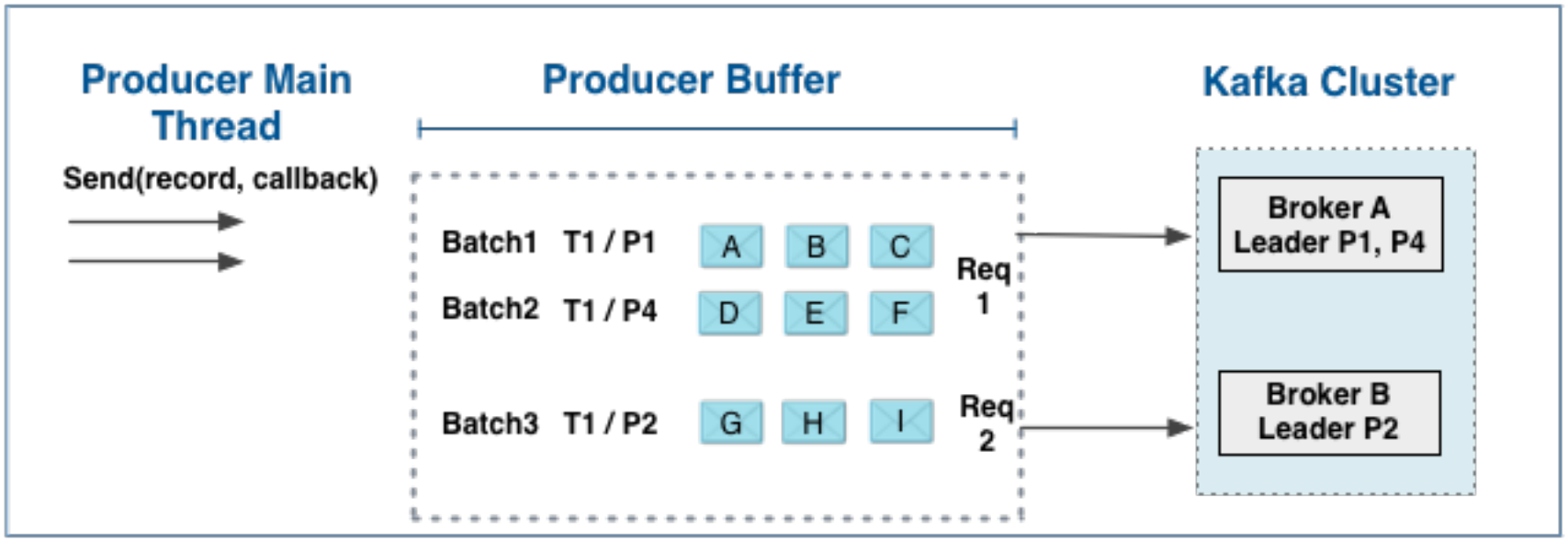
Improving Application Reliability

- Commits
- **Batching**
- Transactions
- Handling Consumer Failure
- Offset Management
- Hands-On Exercise: Detecting and Suppressing Duplicate Messages
- Hands-On Exercise: Handling Invalid Records
- Handling Producer Failure
- Essential Points

Batching

- Producers typically batch records for higher throughput
- Producers have a buffer of memory designated to hold messages

Producer Batches



Producer Batch Configuration

Setting	Description	Default	Recommendation
`batch.size`	Amount of memory in bytes used for each batch of records to the same topic/partition	`16384`	Adjust based on message size and desired throughput
`linger.ms`	Amount of time to wait prior to sending a batch of records	`0`	Adjust based on desired latency
`max.request.size`	Maximum size of a request in bytes sent to a broker	`1048576`	
`buffer.memory`	Total bytes of memory the producer can use to buffer records waiting to be sent.	`32MiB`	Adjust based on producer heap-size

Other Notable Producer Configuration Settings

Setting	Description	Default	Recommendation
`retries`	Number of times producer will retry a failed message, after which an exception will be thrown	`0`	Adjust based on your use-case. NOTE: *Must be a RetriableException*
`retry.backoff.ms`	Milliseconds to wait if a request times out	`60_000`	
`request.timeout.ms`	Maximum number of milliseconds a producer waits to time out a request	`30_000`	
`max.block.ms`	Maximum number of milliseconds for `send()` threads to block if send buffer is full	`60_000`	`Long.MAX_VALUE` (Need to handle exceptions otherwise)

Producer Batch Configuration (Cont'd)

- Requests are made up of multiple batches to the same broker (up to `max.request.size`)
- `linger.ms` can be used to wait for more data to accrue in the buffer prior to sending
- `max.request.size` is limited by `max.message.bytes` for a topic/service
- https://kafka.apache.org/10/javadoc/index.html?org/apache/kafka/clients/producer/KafkaProducer.html

Chapter Topics

Improving Application Reliability

- Commits
- Batching
- **Transactions**
- Handling Consumer Failure
- Offset Management
- Hands-On Exercise: Detecting and Suppressing Duplicate Messages
- Hands-On Exercise: Handling Invalid Records
- Handling Producer Failure
- Essential Points

Transactions and Idempotence

- There is *experimental* support for idempotence and transactions in Kafka
- Cloudera does not currently support these features
- https://cwiki.apache.org/confluence/display/KAFKA/Transactional+Messaging+in+Kafka

Chapter Topics

Improving Application Reliability

- Commits
- Batching
- Transactions
- **Handling Consumer Failure**
- Offset Management
- Hands-On Exercise: Detecting and Suppressing Duplicate Messages
- Hands-On Exercise: Handling Invalid Records
- Handling Producer Failure
- Essential Points

Handling Consumer Failure

- Offsets of messages read by consumers should be tracked
- Consumer offsets provide a way of knowing which messages have been read already
- Kafka provides built-in offset management in the topic `__consumer_offsets`
- Consumers periodically update this topic with offsets of messages they've read
- Kafka provides both manual and automatic offset updates in the consumer API

Demo: Observing Duplicate Messages Caused By Consumer Failure

- **In this demo, you will see how duplicate messages can be received by consumers**
- **Demo directory: `demos/observe-duplicate-messages`**

Chapter Topics

Improving Application Reliability

- Commits
- Batching
- Transactions
- Handling Consumer Failure
- **Offset Management**
- Hands-On Exercise: Detecting and Suppressing Duplicate Messages
- Hands-On Exercise: Handling Invalid Records
- Handling Producer Failure
- Essential Points

Managing Offsets in Consumers

- **Automatic Commits**
  - `enable.auto.commit=true`
  - Background thread commits offsets of messages at `auto.commit.interval.ms`
  - Can give you "at least once" delivery
    - However, consumers must ensure that data returned by `poll()` is consumed prior to the next call to `poll()`
    - And, prior to calling `consumer.close()`

Managing Offsets in Consumers - Manual

- **Managing offsets manually provides more control over committed offsets**
- `enable.auto.commit=false`
  - Developers must call `commitSync()` or `commitAsync()`
  - Code examples at https://kafka.apache.org/10/javadoc/index.html?org/apache/kafka/clients/consumer/KafkaConsumer.html

Chapter Topics

Improving Application Reliability

- Commits
- Batching
- Transactions
- Handling Consumer Failure
- Offset Management
- **Hands-On Exercise: Detecting and Suppressing Duplicate Messages**
- Hands-On Exercise: Handling Invalid Records
- Handling Producer Failure
- Essential Points

Hands-On Exercise: Detecting and Suppressing Duplicate Messages

- **In this exercise, you will see that restarting a Kafka consumer can result in it receiving duplicate messages**
- **Exercise directory: `exercise-code/detect-duplicate-messages`**

Chapter Topics

Improving Application Reliability

- Commits
- Batching
- Transactions
- Handling Consumer Failure
- Offset Management
- Hands-On Exercise: Detecting and Suppressing Duplicate Messages
- **Hands-On Exercise: Handling Invalid Records**
- Handling Producer Failure
- Essential Points

Hands-On Exercise: Handling Invalid Records

- In this exercise, you will practice managing offsets for a topic, which is often used to skip corrupt data that causes Kafka consumers to fail
- Exercise directory: `exercise-code/handling-invalid-records`

Chapter Topics

Improving Application Reliability

- Commits
- Batching
- Transactions
- Handling Consumer Failure
- Offset Management
- Hands-On Exercise: Detecting and Suppressing Duplicate Messages
- Hands-On Exercise: Handling Invalid Records
- **Handling Producer Failure**
- Essential Points

Handling Producer Failure

- **Monitor your brokers to detect changes in throughput**
- **Monitor producer applications for errors**
- **Producer metrics can be collected**
- **Design your application to handle producer failure gracefully**
  - If producer loses contact with its source, data may be lost
  - Restarting producers may result in duplicate messages being sent

Chapter Topics

Improving Application Reliability

- Commits
- Batching
- Transactions
- Handling Consumer Failure
- Offset Management
- Hands-On Exercise: Detecting and Suppressing Duplicate Messages
- Hands-On Exercise: Handling Invalid Records
- Handling Producer Failure
- **Essential Points**

Essential Points

- **Kafka provides reliable storage and retrieval of messages**
- **Producers have options to ensure order and delivery**
- **Consumers have options for ensure reliable reads**

Bibliography

The following offer more information on topics discussed in this chapter

- **Provide intuitive request timeouts (KIP-91)**
  - KIP-91
- **Kafka Producer/Settings**
  - Producer and Consumer Javadocs provide good insight
  - https://kafka.apache.org/10/documentation.html#api
- **Cloudera Documentation**
  - Configuring High Availability and Consistency

cloudera®

Improving Application Scalability

Chapter 8

cloudera®

Course Chapters

- Introduction
- Kafka Overview
- Deploying Apache Kafka
- Kafka Command Line Basics
- Kafka Java API Basics
- Improving Availability through Replication
- Improving Application Reliability
- **Improving Application Scalability**
- Monitoring and Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka Cluster
- Replicating Kafka to Another Cluster
- Conclusion

cloudera

Chapter Topics

Improving Application Scalability

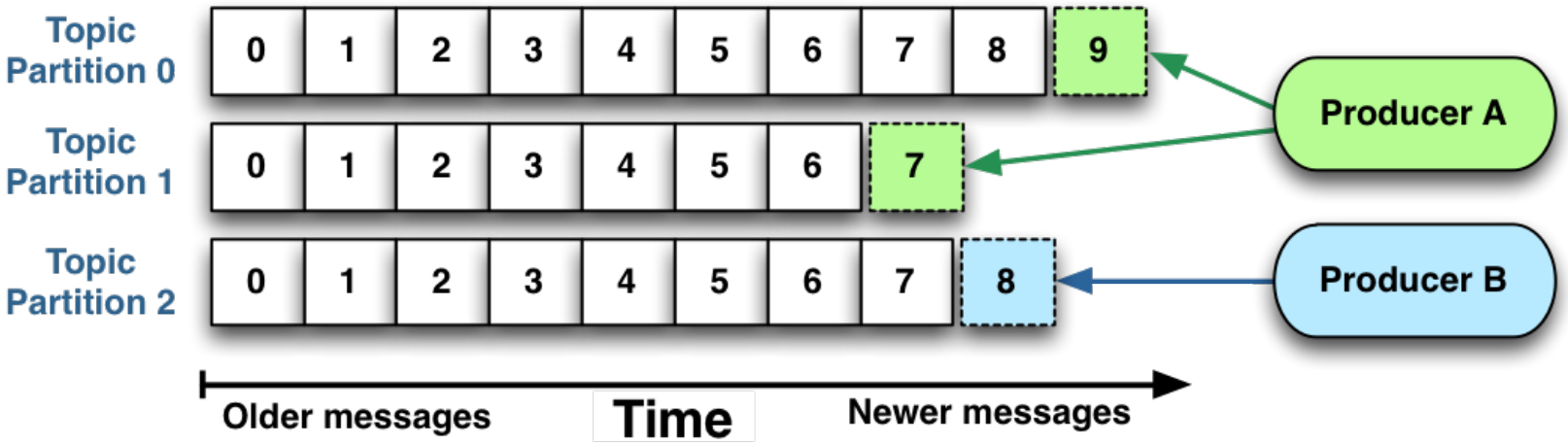
- **Partitioning**
- Hands-On Exercise: Observing How Partitioning Affects Performance
- Consumer Groups
- Hands-On Exercise: Implementing Consumer Groups
- Consumer Rebalancing
- How Messages are Partitioned
- Hands-On Exercise: Using a Key to Control Partition Assignment
- Essential Points

Scaling Kafka

- **Scalability is one of the key benefits of Kafka**
- **Two features let you scale Kafka for performance**
  - Topic partitions
  - Consumer groups

Topic Partitioning

- **Kafka divides each topic into some number of partitions***
  - Topic 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 or Spark

Chapter Topics

Improving Application Scalability

- Partitioning
- **Hands-On Exercise: Observing How Partitioning Affects Performance**
- Consumer Groups
- Hands-On Exercise: Implementing Consumer Groups
- Consumer Rebalancing
- How Messages are Partitioned
- Hands-On Exercise: Using a Key to Control Partition Assignment
- Essential Points

Hands-On Exercise: Observing How Partitioning Affects Performance

- **In this exercise, you will observe how using multiple partitions on a topic changes its performance characteristics**
- **Exercise directory: `exercise-code/partitioning-performance`**

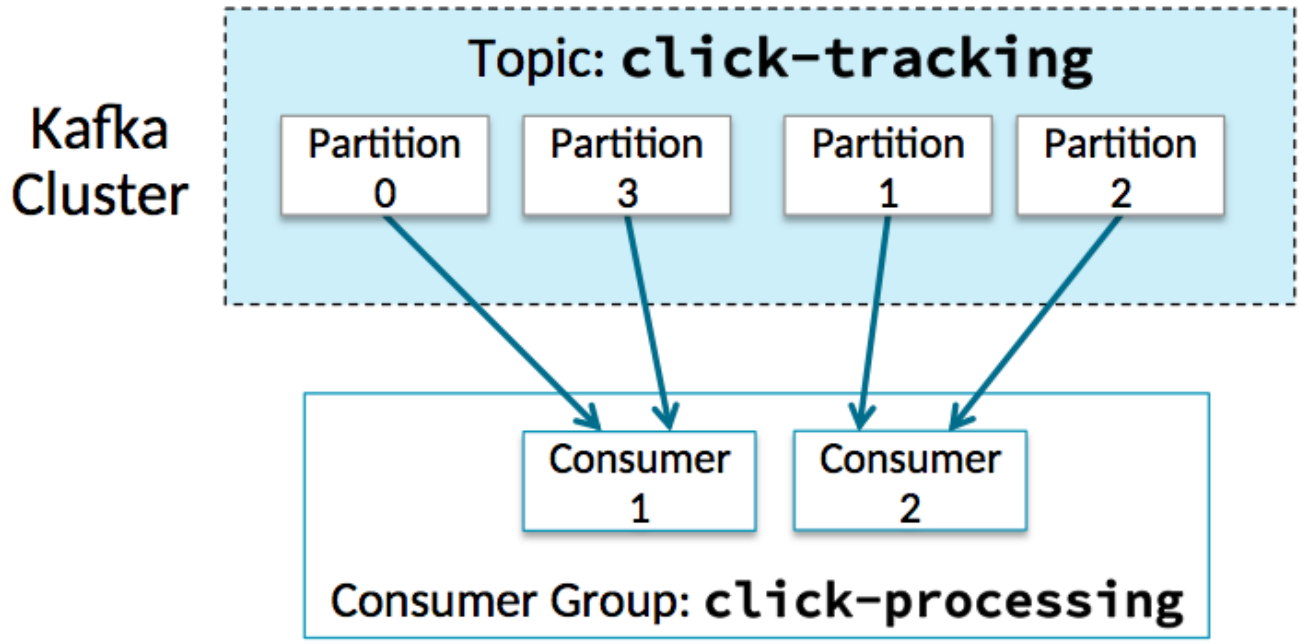
Chapter Topics

Improving Application Scalability

- Partitioning
- Hands-On Exercise: Observing How Partitioning Affects Performance
- **Consumer Groups**
- Hands-On Exercise: Implementing Consumer Groups
- Consumer Rebalancing
- How Messages are Partitioned
- Hands-On Exercise: Using a Key to Control Partition Assignment
- Essential Points

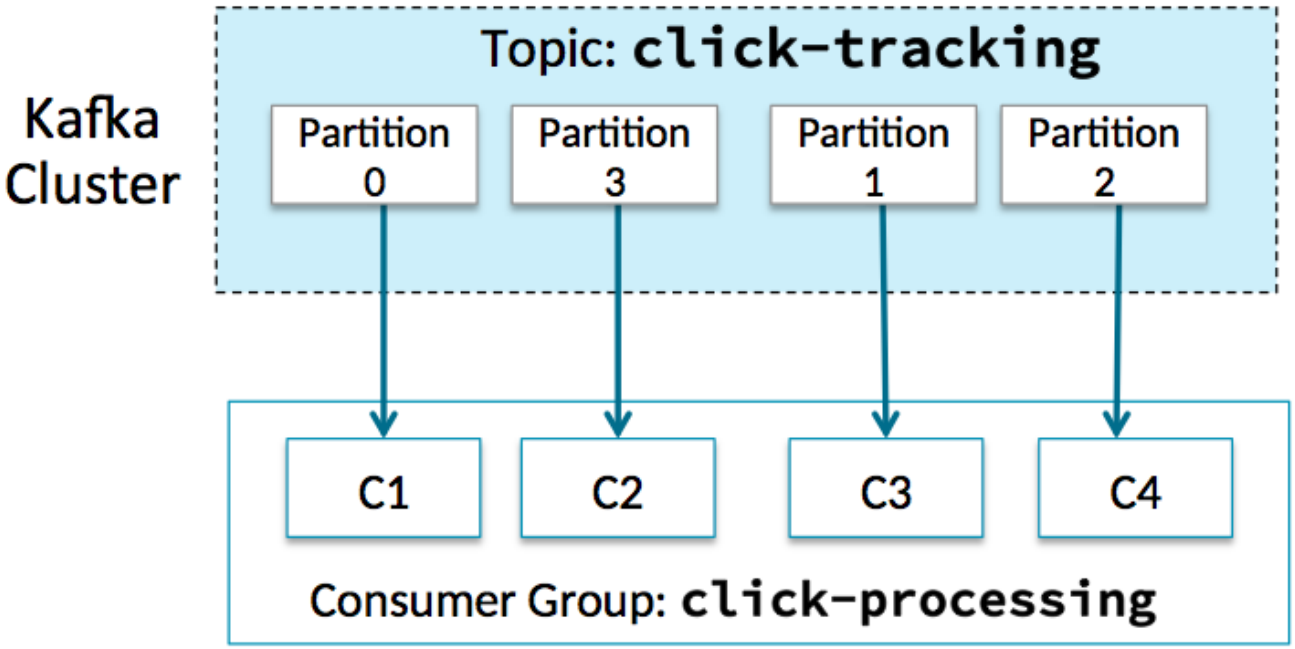
Consumer Groups

- One or more consumers can form their own consumer group that work together to consume the messages in a topic
- Each partition is consumed by only one member of a consumer group
- Message ordering is preserved per partition, but not across the topic



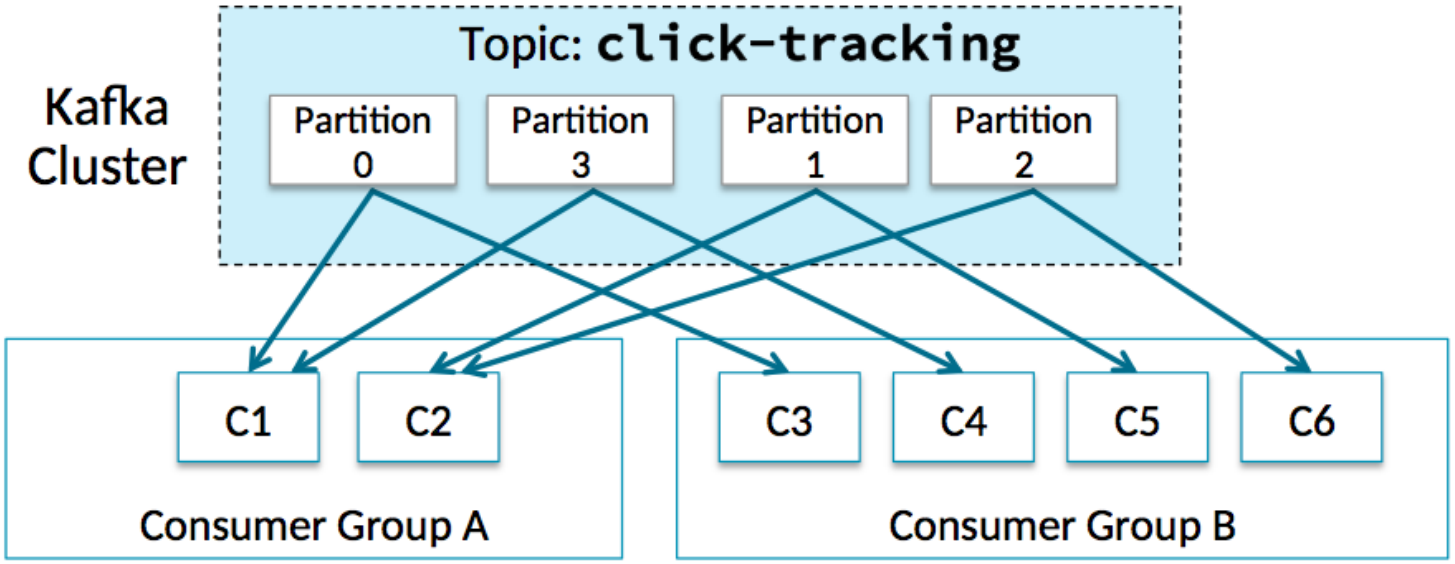
Increasing Consumer Throughput

- Additional consumers can be added to scale consumer group processing
- Consumer instances that belong to the same consumer group can be in separate processes or on separate machines



Multiple Consumer Groups

- **Each message published to a topic is delivered to one consumer instance within each subscribing consumer group**
- **Kafka scales to large numbers of consumer groups and consumers**



Chapter Topics

Improving Application Scalability

- Partitioning
- Hands-On Exercise: Observing How Partitioning Affects Performance
- Consumer Groups
- **Hands-On Exercise: Implementing Consumer Groups**
- Consumer Rebalancing
- How Messages are Partitioned
- Hands-On Exercise: Using a Key to Control Partition Assignment
- Essential Points

Hands-On Exercise: Implementing Consumer Groups

- In this exercise, you will practice setting up a consumer group, which allows multiple consumers to coordinate message processing for a given topic
- Exercise directory: `exercise-code/consumer-groups`

Chapter Topics

Improving Application Scalability

- Partitioning
- Hands-On Exercise: Observing How Partitioning Affects Performance
- Consumer Groups
- Hands-On Exercise: Implementing Consumer Groups
- **Consumer Rebalancing**
- How Messages are Partitioned
- Hands-On Exercise: Using a Key to Control Partition Assignment
- Essential Points

Chapter Topics

Improving Application Scalability

- Partitioning
- Hands-On Exercise: Observing How Partitioning Affects Performance
- Consumer Groups
- Hands-On Exercise: Implementing Consumer Groups
- Consumer Rebalancing
- **How Messages are Partitioned**
- Hands-On Exercise: Using a Key to Control Partition Assignment
- Essential Points

Chapter Topics

Improving Application Scalability

- Partitioning
- Hands-On Exercise: Observing How Partitioning Affects Performance
- Consumer Groups
- Hands-On Exercise: Implementing Consumer Groups
- Consumer Rebalancing
- How Messages are Partitioned
- **Hands-On Exercise: Using a Key to Control Partition Assignment**
- Essential Points

Hands-On Exercise: Using a Key to Control Partition Assignment

- **In this exercise, you will learn how Kafka assigns messages to partitions, and how you can influence that assignment**
- **Exercise directory: `exercise-code/control-partitioning`**

Chapter Topics

Improving Application Scalability

- Partitioning
- Hands-On Exercise: Observing How Partitioning Affects Performance
- Consumer Groups
- Hands-On Exercise: Implementing Consumer Groups
- Consumer Rebalancing
- How Messages are Partitioned
- Hands-On Exercise: Using a Key to Control Partition Assignment
- **Essential Points**

Bibliography

The following offer more information on topics discussed in this chapter

- **Cloudera blog on consumer groups**
  - Scalability of Consumer Groups
- **Cloudera Enterprise Documentation on Consumer Groups**
  - Clients - Consumer Groups and Fetching

cloudera®

Monitoring and Managing Kafka

Chapter 9

cloudera®

Course Chapters

- Introduction
- Kafka Overview
- Deploying Apache Kafka
- Kafka Command Line Basics
- Kafka Java API Basics
- Improving Availability through Replication
- Improving Application Reliability
- Improving Application Scalability
- **Monitoring and Managing Kafka**
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka Cluster
- Replicating Kafka to Another Cluster
- Conclusion

cloudera

Chapter Topics

Monitoring and Managing Kafka

- **Charts and Metrics in Cloudera Manager**
- Service Logs
- Upgrading Kafka
- Diagnosing Service Failure
- Hands-On Exercise: Monitoring and Managing Kafka
- Hands-On Exercise: Troubleshooting Kafka Topics
- Essential Points

Kafka Charts and Metrics in Cloudera Manager

- **We will review Kafka charts and metrics during the exercise**

Chapter Topics

Monitoring and Managing Kafka

- Charts and Metrics in Cloudera Manager
- **Service Logs**
- Upgrading Kafka
- Diagnosing Service Failure
- Hands-On Exercise: Monitoring and Managing Kafka
- Hands-On Exercise: Troubleshooting Kafka Topics
- Essential Points

Kafka Service Logs

- **Logs are the first line of investigation for problems**
- **We will review how to obtain Kafka logs in the exercise**

Chapter Topics

Monitoring and Managing Kafka

- Charts and Metrics in Cloudera Manager
- Service Logs
- **Upgrading Kafka**
- Diagnosing Service Failure
- Hands-On Exercise: Monitoring and Managing Kafka
- Hands-On Exercise: Troubleshooting Kafka Topics
- Essential Points

Cloudera Notes Regarding Upgrades

- Installing and Upgrading CDK
- **Please see "Client/Broker Compatibility" section here:**
  - Kafka Multiple Versions

Chapter Topics

Monitoring and Managing Kafka

- Charts and Metrics in Cloudera Manager
- Service Logs
- Upgrading Kafka
- **Diagnosing Service Failure**
- Hands-On Exercise: Monitoring and Managing Kafka
- Hands-On Exercise: Troubleshooting Kafka Topics
- Essential Points

Diagnosing Service Failure

- **Broker Logs**
- **Health Checks (Available in CM)**

Chapter Topics

Monitoring and Managing Kafka

- Charts and Metrics in Cloudera Manager
- Service Logs
- Upgrading Kafka
- Diagnosing Service Failure
- **Hands-On Exercise: Monitoring and Managing Kafka**
- Hands-On Exercise: Troubleshooting Kafka Topics
- Essential Points

Hands-On Exercise: Monitoring and Managing Kafka

- **In this exercise, you will set up monitoring for important metrics in Cloudera Manager.**
- **Exercise directory: `exercise-code/monitoring-managing`**

Chapter Topics

Monitoring and Managing Kafka

- Charts and Metrics in Cloudera Manager
- Service Logs
- Upgrading Kafka
- Diagnosing Service Failure
- Hands-On Exercise: Monitoring and Managing Kafka
- **Hands-On Exercise: Troubleshooting Kafka Topics**
- Essential Points

Hands-On Exercise: Troubleshooting Kafka Topics

- **In this exercise, you will troubleshoot problems with topic assignments and utilization.**
- **Exercise directory: `exercise-code/troubleshooting-kafka-topics`**

Chapter Topics

Monitoring and Managing Kafka

- Charts and Metrics in Cloudera Manager
- Service Logs
- Upgrading Kafka
- Diagnosing Service Failure
- Hands-On Exercise: Monitoring and Managing Kafka
- Hands-On Exercise: Troubleshooting Kafka Topics
- **Essential Points**

Bibliography

The following offer more information on topics discussed in this chapter

- **Kafka Metrics from Cloudera Manager**
  - http://tiny.cloudera.com/kafch09a1

cloudera®

Message Structure, Format, and Versioning

Chapter 10

cloudera®

Course Chapters

- Introduction
- Kafka Overview
- Deploying Apache Kafka
- Kafka Command Line Basics
- Kafka Java API Basics
- Improving Availability through Replication
- Improving Application Reliability
- Improving Application Scalability
- Monitoring and Managing Kafka
- **Message Structure, Format, and Versioning**
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka Cluster
- Replicating Kafka to Another Cluster
- Conclusion

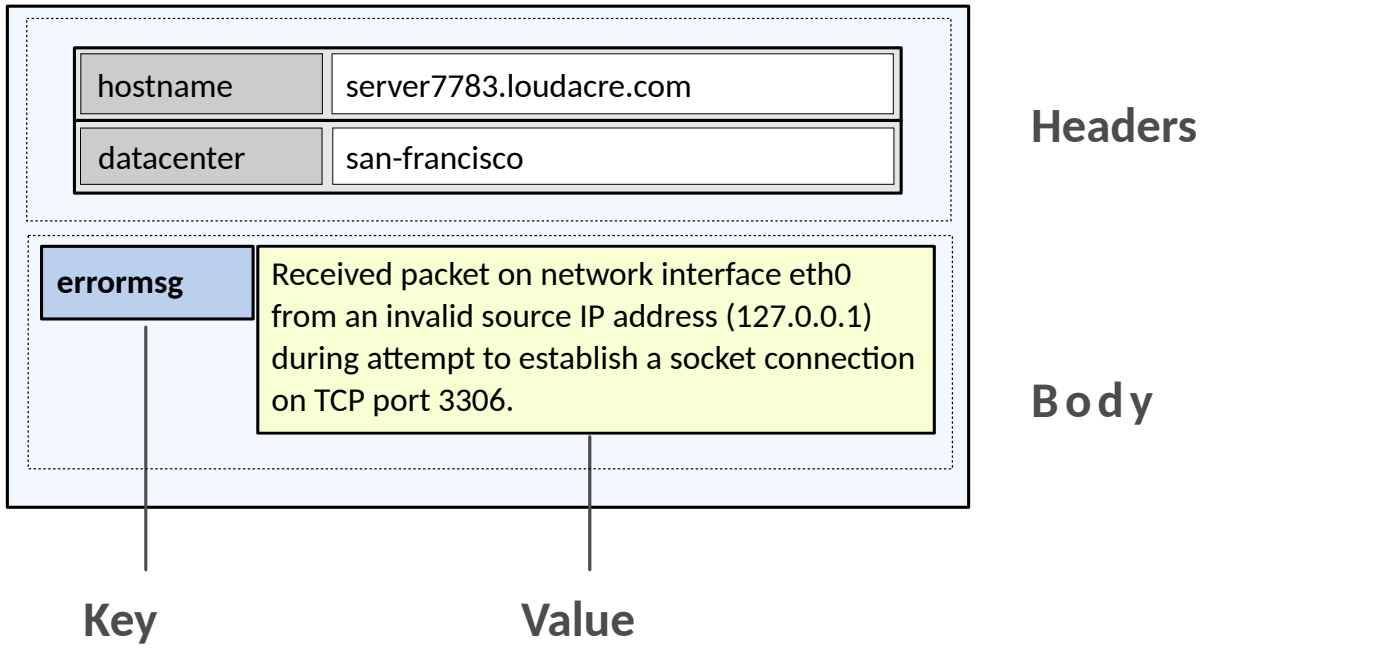
Chapter Topics

Message Structure, Format, and Versioning

- **Message Structure**
- Message Efficiency
- Hands-On Exercise: Understanding How Message Size Affects Performance
- Message Compatibility
- Apache Avro Overview
- Avro Schemas
- Hands-On Exercise: Working with Avro
- Hands-On Exercise: Designing and Using an Avro Schema with Kafka
- Schema Evolution
- Schema Registry and Versioning
- Essential Points

Anatomy of a Kafka Record

- **Headers are an optional set of name-value pairs**
  - Typically used for metadata and message processing
  - Name is of type `String`, value is a `byte[]`
- **The payload is sent in the record's body**
  - Consists of a key (often empty) and value
  - Both can be of any type, but each is ultimately converted to/from `byte[]`



Using Keys, Values, and Headers

- **Key and value are set in constructor of `ProducerRecord`**
  - Accessed in `ConsumerRecord` with `key()` and `value()` methods
- **Headers can be set in constructor of `ProducerRecord`**
  - Accessed in `ConsumerRecord` by calling `headers()` method

Chapter Topics

Message Structure, Format, and Versioning

- Message Structure
- **Message Efficiency**
- Hands-On Exercise: Understanding How Message Size Affects Performance
- Message Compatibility
- Apache Avro Overview
- Avro Schemas
- Hands-On Exercise: Working with Avro
- Hands-On Exercise: Designing and Using an Avro Schema with Kafka
- Schema Evolution
- Schema Registry and Versioning
- Essential Points

Data Serialization

- ***Serialization* refers to how we represent data in memory as a series of bytes**
  - Allows us to save data to disk or send it across the network
  - *Deserialization* allows us to read that data back into memory
- **For example, consider how to serialize the number 108125150**
  - 9 bytes when stored as a Java `String`
  - 4 bytes when stored as a Java `int`
  - This distinction can make a major difference in efficiency and scalability
- **Many programming languages and libraries support serialization**
  - Such as `Serializable` in Java or `pickle` in Python
  - However, backwards compatibility and cross-language support can be challenging

Kafka Serializers and Deserializers

- **The `org.apache.kafka.common.serialization` package defines two important interfaces**
  - `Serializer<T>`
    - Used by the producer
    - Converts an object of a specified type into a `byte[]`
  - `Deserializer<T>`
    - Used by the consumer
    - Converts the `byte[]` back into the original object
- **Specify the implementation's class name via the client's properties**
  - Key and value serializers are configured independently

Chapter Topics

Message Structure, Format, and Versioning

- Message Structure
- Message Efficiency
- **Hands-On Exercise: Understanding How Message Size Affects Performance**
- Message Compatibility
- Apache Avro Overview
- Avro Schemas
- Hands-On Exercise: Working with Avro
- Hands-On Exercise: Designing and Using an Avro Schema with Kafka
- Schema Evolution
- Schema Registry and Versioning
- Essential Points

Hands-On Exercise: Understanding How Message Size Affects Performance

- **In this exercise, you will compare how message size affects throughput. You will also learn how to improve performance through more efficient custom serialization.**
- **Exercise directory: `exercise-code/message-size-performance`**

Chapter Topics

Message Structure, Format, and Versioning

- Message Structure
- Message Efficiency
- Hands-On Exercise: Understanding How Message Size Affects Performance
- **Message Compatibility**
- Apache Avro Overview
- Avro Schemas
- Hands-On Exercise: Working with Avro
- Hands-On Exercise: Designing and Using an Avro Schema with Kafka
- Schema Evolution
- Schema Registry and Versioning
- Essential Points

Consider Compatibility When Changing Message Content

- **One of Kafka's key benefits is decoupling of producers and consumers**
- **These may be part of different applications, written by different teams**
- **Changing format of a producer's messages can cause consumers to fail**
  - Even worse is if consumers don't fail, but misinterpret data
  - Communicate and test your plan to maintain (or break) compatibility

Chapter Topics

Message Structure, Format, and Versioning

- Message Structure
- Message Efficiency
- Hands-On Exercise: Understanding How Message Size Affects Performance
- Message Compatibility
- **Apache Avro Overview**
- Avro Schemas
- Hands-On Exercise: Working with Avro
- Hands-On Exercise: Designing and Using an Avro Schema with Kafka
- Schema Evolution
- Schema Registry and Versioning
- Essential Points

What is Apache Avro?

- **Avro is an efficient data serialization framework**
  - Top-level Apache project created by Doug Cutting (creator of Hadoop)
  - Widely supported throughout Hadoop and its ecosystem
- **Offers compatibility without sacrificing performance**
  - Data is serialized according to a *schema* you define
  - Read/write data in Java, C, C++, C#, Python, PHP, and other languages
  - Serializes data using a highly-optimized binary encoding
  - Specifies rules for *evolving* your schema over time

Chapter Topics

Message Structure, Format, and Versioning

- Message Structure
- Message Efficiency
- Hands-On Exercise: Understanding How Message Size Affects Performance
- Message Compatibility
- Apache Avro Overview
- **Avro Schemas**
- Hands-On Exercise: Working with Avro
- Hands-On Exercise: Designing and Using an Avro Schema with Kafka
- Schema Evolution
- Schema Registry and Versioning
- Essential Points

Basic Schema Example

- **Excerpt from a SQL CREATE TABLE statement**

```
CREATE TABLE employees
(id INT, name STRING, title STRING, bonus INT)
```

- **Equivalent Avro schema**

```
{
"namespace": "com.loudacre.data",
"type": "record",
"name": "Employee",
"fields": [
    {"name": "id", "type": "int"},
    {"name": "name", "type": "string"},
    {"name": "title", "type": "string"},
    {"name": "bonus", "type": "int"}
  ]
}
```

Avro Schemas

- **Avro schemas define the structure of your data**
  - Similar in concept to `CREATE TABLE` in SQL, but more flexible
  - Schemas are represented using JSON
- **Three approaches for mapping a Java object to a schema**
  - **Generic**: Write code to map each schema field to a field in your object
    - Flexible, but sacrifices compile-time type safety
  - **Reflect**: Generate a schema from an existing class
    - Easy, but slower at runtime and may limit compatibility
  - **Specific**: Generate a Java class from your schema
    - Recommended, since it offers best performance and compatibility
    - There is even a Maven goal for doing this as part of your build

Supported Types in Avro Schemas (Simple)

- **A simple type holds exactly one value**
- **Avro's `string` type can be mapped to an Avro utility class, if desired**
  - Using `org.apache.avro.util.Utf8` may improve performance

Name	Description	Java Equivalent
`null`	An absence of a value	`null`
`boolean`	A binary value	`boolean`
`int`	32-bit signed integer	`int`
`long`	64-bit signed integer	`long`
`float`	Single-precision floating point value	`float`
`double`	Double-precision floating point value	`double`
`bytes`	Sequence of 8-bit unsigned bytes	`java.nio.ByteBuffer`
`string`	Sequence of Unicode characters	`java.lang.CharSequence`

Supported Types in Avro Schemas (Complex)

- **Avro also supports six complex types**

Name	Description
`record`	A user-defined type composed of one or more named fields
`enum`	A specified set of values
`array`	Zero or more values of the same type
`map`	Set of key-value pairs; key is `string` while value is of specified type
`union`	Exactly one value matching a specified set of types
`fixed`	A fixed number of 8-bit unsigned bytes

- **The `record` type is the most important of these**
  - The other types are primarily used to define a record's fields

Specifying Default Values in the Schema

- **The SQL CREATE TABLE statement allows you to specify a default value**
  - Used when no value was explicitly set for a field
- **Avro also supports setting a default value in the schema**

```
{"namespace":"com.loudacre.data",
"type":"record",
"name":"Invoice",
"fields":[
    {"name":"id", "type":"int"},
    {"name":"taxcode", "type":"int", "default": 39},
    {"name":"lang", "type":"string", "default": "EN_US"}
  ]
}
```

Avro Schemas and Null Values

- **Avro checks for null values when serializing the data**
- **Null values are only allowed *when explicitly specified* in the schema**
  - The title and bonus fields in the example below allow null values
  - If using a default, its type must be listed first in the union

```
{"namespace":"com.loudacre.data",
  "type": "record",
  "name": "Employee",
  "fields": [
    {"name": "id", "type": "int"},
    {"name": "name", "type": "string"},
    {"name": "title", "type": ["null", "string"]},
    {"name": "bonus", "type": ["null", "int"]}
}
```

Schema Example with Complex Types

- **The following example shows a record with an enum and a string array**

```
{"namespace":"com.loudacre.data",
  "type":"record",
  "name":"Ringtone",
  "fields":[
    {"name":"id", "type":"int"},
    {"name":"title", "type":"string"},
    {"name":"price", "type":"int"},
    {"name":"format", "type": {
     "name":"RingtoneFormat", "type":"enum",
     "symbols":["MP3", "AAC", "MIDI"]}
    },
    {"name":"tags", "type": {
  "type":"array", "items":"string"}
    }
}
```

Documenting Your Schema

- **It's a good practice to document any ambiguities in a schema**
  - All types (including record) support an optional `doc` attribute

```
{"namespace":"com.loudacre.data",
  "type":"record",
  "name":"WebProduct",
  "doc":"Item currently sold in Loudacre's online store",
  "fields":[
    {"name":"id", "type":"int", "doc": "Product SKU"},
    {"name":"shipwt", "type":"int",
   "doc": "Shipping weight, in pounds"},
    {"name":"price", "type":"int",
   "doc": "Retail price, in cents (US)"}
   ]
}
```

Avro Container Format

- **Avro also defines a container file format for storing Avro records**
  - Also known as “Avro data file format”
  - Cross-language support for reading and writing data
- **Supports compressing *blocks* (groups) of records**
  - It is “splittable” for efficient processing by MapReduce
- **This format is self-describing**
  - Each file contains a copy of the schema used to write its data
  - All records in a file must use the same schema

Inspecting Avro Data Files with Avro Tools

- **Avro data files are an efficient way to store data**
  - However, the binary format makes debugging difficult
- **The Avro package offers an `avro-tools` command**
  - This tool allows you to read the schema or data for an Avro file

```
$ avro-tools tojson mydatafile.avro
{"name":"Alice","salary": 56500,"city":"Anaheim"}
{"name":"Bob","salary": 51400,"city":"Bellevue"}
```

```
$ avro-tools getschema mydatafile.avro
{
"type" : "record",
"name" : "DeviceData",
"namespace" : "com.loudacre.data",  ...rest of schema follows
```

Aside: Generating Java Source from an Avro Schema

- **When using Avro "specific" records, you must first generate the Java code**
- **One way to generate code is to use the `avro-tools` command**
  - Output directory must already exist
  - We recommend that you generate code to its own directory

```
$ mkdir generated-src

$ avro-tools compile schema employee.avsc generated-src
```

- **Using Maven to generate Java code from a schema is even easier**
  - You will do this in an upcoming hands-on exercise
- **You may notice that the fields in the generated code are deprecated**
  - This is to encourage you to use their `get` and `set` methods instead

Chapter Topics

Message Structure, Format, and Versioning

- Message Structure
- Message Efficiency
- Hands-On Exercise: Understanding How Message Size Affects Performance
- Message Compatibility
- Apache Avro Overview
- Avro Schemas
- **Hands-On Exercise: Working with Avro**
- Hands-On Exercise: Designing and Using an Avro Schema with Kafka
- Schema Evolution
- Schema Registry and Versioning
- Essential Points

Hands-On Exercise: Working with Avro

- **In this exercise, you will practice writing an Avro schema, generating Java code from that schema, writing Avro data to a file, and then viewing that file's data and schema**
- **Exercise directory: `exercise-code/working-with-avro`**

Chapter Topics

Message Structure, Format, and Versioning

- Message Structure
- Message Efficiency
- Hands-On Exercise: Understanding How Message Size Affects Performance
- Message Compatibility
- Apache Avro Overview
- Avro Schemas
- Hands-On Exercise: Working with Avro
- **Hands-On Exercise: Designing and Using an Avro Schema with Kafka**
- Schema Evolution
- Schema Registry and Versioning
- Essential Points

Hands-On Exercise: Designing and Using an Avro Schema with Kafka

- **In this exercise, you will design an Avro schema, configure a producer to use it for serializing data sent to a topic, and configure a consumer to use it when reading data from the topic**
- **Exercise directory: `exercise-code/avro-schema-kafka`**

Chapter Topics

Message Structure, Format, and Versioning

- Message Structure
- Message Efficiency
- Hands-On Exercise: Understanding How Message Size Affects Performance
- Message Compatibility
- Apache Avro Overview
- Avro Schemas
- Hands-On Exercise: Working with Avro
- Hands-On Exercise: Designing and Using an Avro Schema with Kafka
- **Schema Evolution**
- Schema Registry and Versioning
- Essential Points

Schema Evolution

- **The structure of your data will change over time**
  - Fields may be added, removed, changed, or renamed
  - In SQL, these are handled with `ALTER TABLE` statements
- **These changes can break compatibility with many formats**
- **Data written to Avro data files is always readable**
  - The schema used to write the data is embedded in the file itself
  - However, an application reading data might expect the *new* structure
- **Avro has a unique approach to maintaining forward compatibility**
  - A reader can use a different schema than the writer

Schema Evolution: A Practical Example (1)

- **Imagine that we have written millions of records with this schema**

```
{"namespace": "com.loudacre.data",
 "type": "record",
 "name": "CustomerContact",
 "fields": [
   {"name": "id", "type": "int"},
   {"name": "name", "type": "string"},
   {"name": "faxNumber", "type": "string"}
]}
```

Schema Evolution: A Practical Example (2)

- **We would like to update this based on the schema below**
  - Renamed `id` field to `customerId`
  - Changed type from `int` to `long`
  - Added `email` field

```
{"namespace": "com.loudacre.data",
 "type": "record",
 "name": "CustomerContact",
 "fields": [
   {"name": "customerId", "type": "long"},
   {"name": "name", "type": "string"},
   {"name": "faxNumber", "type": "string"},
   {"name": "email", "type": "string"}
]}
```

Schema Evolution: A Practical Example (3)

- **We could use the new schema to write new data**
  - Applications that use the new schema could read the new data
- **Unfortunately, new applications wouldn't be able to read the *old* data**
  - We must make a few schema changes to improve compatibility

Schema Evolution: A Practical Example (4)

- **If you rename a field, you must specify an alias for the old name(s)**
  - Here, we map the old `id` field to the new `customerId` field

```
{"namespace": "com.loudacre.data",
 "type": "record",
 "name": "CustomerContact",
 "fields": [
   {"name": "customerId", "type": "long",
            "aliases": ["id"]},
   {"name": "name", "type": "string"},
   {"name": "faxNumber", "type": "string"},
   {"name": "email", "type": "string"}
]}
```

Schema Evolution: A Practical Example (5)

- **Newly-added fields will lack values for records previously written**
  - You must specify a default value
  - In this case, the field is made nullable so `null` can be the default

```
{"namespace": "com.loudacre.data",
 "type": "record",
 "name": "CustomerContact",
 "fields": [
   {"name": "customerId", "type": "long",
            "aliases": ["id"]},
   {"name": "name", "type": "string"},
   {"name": "faxNumber", "type": "string"},
   {"name": "email", "type": ["null", "string"],
            "default": null}
]}
```

Schema Evolution: Compatible Changes

- **The following changes will not affect existing readers**
  - Adding, changing, or removing a `doc` attribute
  - Changing a field's default value
  - Adding a new field with a default value
  - Removing a field that specified a default value
  - Promoting a field to a wider type (e.g., `int` to `long`)
  - Adding aliases for a field

Schema Evolution: Incompatible Changes

- **The following are some changes that may break compatibility**
  - Changing the record's `name` or `namespace` attributes
  - Adding a new field without a default value
  - Removing a symbol from an `enum`
  - Removing a type from a `union`
  - Modification to a field's type that could result in truncation
- **Such changes may require you to perform the following steps**
  - Read your old data (using the original schema)
  - Modify data as needed in your application
  - Write the new data (using the new schema)
  - Existing readers/writers may need to be updated to use new schema

Chapter Topics

Message Structure, Format, and Versioning

- Message Structure
- Message Efficiency
- Hands-On Exercise: Understanding How Message Size Affects Performance
- Message Compatibility
- Apache Avro Overview
- Avro Schemas
- Hands-On Exercise: Working with Avro
- Hands-On Exercise: Designing and Using an Avro Schema with Kafka
- Schema Evolution
- **Schema Registry and Versioning**
- Essential Points

Chapter Topics

Message Structure, Format, and Versioning

- Message Structure
- Message Efficiency
- Hands-On Exercise: Understanding How Message Size Affects Performance
- Message Compatibility
- Apache Avro Overview
- Avro Schemas
- Hands-On Exercise: Working with Avro
- Hands-On Exercise: Designing and Using an Avro Schema with Kafka
- Schema Evolution
- Schema Registry and Versioning
- **Essential Points**

Essential Points

- **The body of Kafka messages consists of a key and value**
  - Both are sent and stored as an array of bytes
  - Producers serialize objects into byte arrays
  - Consumers deserialize byte arrays into objects
  - Kafka provides serializers/deserializers for a few common types
  - You can often improve efficiency by writing custom serialization code
- **Avro is an efficient data serialization framework**
  - Performs serialization based on a schema you define
  - Good choice for maintaining compatibility despite evolution

Bibliography

The following offer more information on topics discussed in this chapter

- **Javadoc for Kafka `serialization` Package**
  - http://tiny.cloudera.com/kafch10a
- **Apache Avro: Getting Started Guide (Java)**
  - http://tiny.cloudera.com/kafch10b
- **Blog Post Describing Avro Schema Registry for Kafka (part 1 of 3)**
  - http://tiny.cloudera.com/kafch10c
- **Blog Post Describing Avro Schema Registry for Kafka (part 2 of 3)**
  - http://tiny.cloudera.com/kafch10d
- **Blog Post Describing Avro Schema Registry for Kafka (part 3 of 3)**
  - http://tiny.cloudera.com/kafch10e

cloudera®

Improving Application Performance

Chapter 11

cloudera®

Course Chapters

- Introduction
- Kafka Overview
- Deploying Apache Kafka
- Kafka Command Line Basics
- Kafka Java API Basics
- Improving Availability through Replication
- Improving Application Reliability
- Improving Application Scalability
- Monitoring and Managing Kafka
- Message Structure, Format, and Versioning
- **Improving Application Performance**
- Improving Kafka Service Performance
- Securing the Kafka Cluster
- Replicating Kafka to Another Cluster
- Conclusion

Chapter Topics

Improving Application Performance

- **Message Size**
- Compression
- Hands-On Exercise: Observing How Compression Affects Performance
- Performance Tuning Strategies for the Developer
- Essential Points

Message Size

- **Please see the following links:**
  - Handling Large Messages
  - What's a good size of a Kafka record

Chapter Topics

Improving Application Performance

- Message Size
- **Compression**
- Hands-On Exercise: Observing How Compression Affects Performance
- Performance Tuning Strategies for the Developer
- Essential Points

Compression

- **We will review options for compression during the exercise**

Chapter Topics

Improving Application Performance

- Message Size
- Compression
- **Hands-On Exercise: Observing How Compression Affects Performance**
- Performance Tuning Strategies for the Developer
- Essential Points

Hands-On Exercise: Observing How Compression Affects Performance

- **In this exercise, you will compare the performance and storage characteristics of four compression codecs by applying them different types of data**
- **Exercise directory: `exercise-code/compression-performance`**

Chapter Topics

Improving Application Performance

- Message Size
- Compression
- Hands-On Exercise: Observing How Compression Affects Performance
- **Performance Tuning Strategies for the Developer**
- Essential Points

Performance Tuning Strategies for the Developer

- Typically involves a balance of latency versus throughput
- Batch sizes
- Linger ms (batch waiting period)

Chapter Topics

Improving Application Performance

- Message Size
- Compression
- Hands-On Exercise: Observing How Compression Affects Performance
- Performance Tuning Strategies for the Developer
- **Essential Points**

cloudera®

Improving Kafka Service Performance

Chapter 12

cloudera®

Course Chapters

- Introduction
- Kafka Overview
- Deploying Apache Kafka
- Kafka Command Line Basics
- Kafka Java API Basics
- Improving Availability through Replication
- Improving Application Reliability
- Improving Application Scalability
- Monitoring and Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- **Improving Kafka Service Performance**
- Securing the Kafka Cluster
- Replicating Kafka to Another Cluster
- Conclusion

Chapter Topics

Improving Kafka Service Performance

- **Cluster Sizing**
- Hands-On Exercise: Planning Capacity Needed for a Use Case
- Identifying Performance Bottlenecks
- Performance Tuning Strategies for the Administrator
- Essential Points

Chapter Topics

Improving Kafka Service Performance

- Cluster Sizing
- **Hands-On Exercise: Planning Capacity Needed for a Use Case**
- Identifying Performance Bottlenecks
- Performance Tuning Strategies for the Administrator
- Essential Points

Hands-On Exercise: Planning Capacity Needed for a Use Case

- **In this exercise, you will determine the suggested cluster size for a given use case, based on details specified by your instructor**
- **Exercise directory: `exercise-code/capacity-planning`**

Chapter Topics

Improving Kafka Service Performance

- Cluster Sizing
- Hands-On Exercise: Planning Capacity Needed for a Use Case
- **Identifying Performance Bottlenecks**
- Performance Tuning Strategies for the Administrator
- Essential Points

Chapter Topics

Improving Kafka Service Performance

- Cluster Sizing
- Hands-On Exercise: Planning Capacity Needed for a Use Case
- Identifying Performance Bottlenecks
- **Performance Tuning Strategies for the Administrator**
- Essential Points

Chapter Topics

Improving Kafka Service Performance

- Cluster Sizing
- Hands-On Exercise: Planning Capacity Needed for a Use Case
- Identifying Performance Bottlenecks
- Performance Tuning Strategies for the Administrator
- **Essential Points**

Bibliography

The following offer more information on topics discussed in this chapter

- **Cloudera's Kafka Deployment Guide**
  - http://tiny.cloudera.com/kafch12a

cloudera®

Securing the Kafka Cluster

Chapter 13

cloudera®

Course Chapters

- Introduction
- Kafka Overview
- Deploying Apache Kafka
- Kafka Command Line Basics
- Kafka Java API Basics
- Improving Availability through Replication
- Improving Application Reliability
- Improving Application Scalability
- Monitoring and Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance
- **Securing the Kafka Cluster**
- Replicating Kafka to Another Cluster
- Conclusion

Chapter Topics

Securing the Kafka Cluster

- **Encryption**
- Authentication
- Authorization
- Auditing
- Essential Points

Encryption Overview

- Encryption protects the confidentiality of data
- Can use it to protect data during network transmission (in motion)
- Can use it to protect data stored on disk (at rest)

Encryption for Data in Motion

- **In Kafka, this is done with SSL (TLS)**
- **Provides confidentiality for underlying network transport protocol**
  - Handshake negotiates cipher and establishes a session key
  - Uses that key to encrypt data for remainder of session
- **Server is identified by a digital certificate**
  - A *Certificate Authority* (CA) validates identity
  - Clients may *also* have certificates
  - CA certs are kept in a *truststore* and used to validate *chain of trust*
- **In Kafka, SSL can have a major performance penalty**
  - For this reason, it's often disabled for inter-broker communication
  - Brokers can simultaneously support both SSL and non-SSL traffic

SSL Configuration Overview

- **Protocols: `PLAINTEXT` (SSL disabled, port 9092) or `SSL` (SSL enabled, port 9093)**
- **Brokers can simultaneously support both SSL and non-SSL traffic**
- **Inter-broker communication protocol is configured independently from clients**
- **Clients specify configuration via properties (protocol, keystore, truststore, etc.)**

Encryption for Data at Rest

- **Can use encryption to protect Kafka's data directory**
- **Cloudera recommends Cloudera Navigator Encrypt for this**
  - Provides transparent encryption for *local* data
  - Used to protect directories containing sensitive data
    - Log files
    - Application data and databases
    - Temporary files created during processing
  - Relies on Cloudera Navigator Key Trustee Server for key storage
  - Kernel module intercepts I/O calls to protected data

Chapter Topics

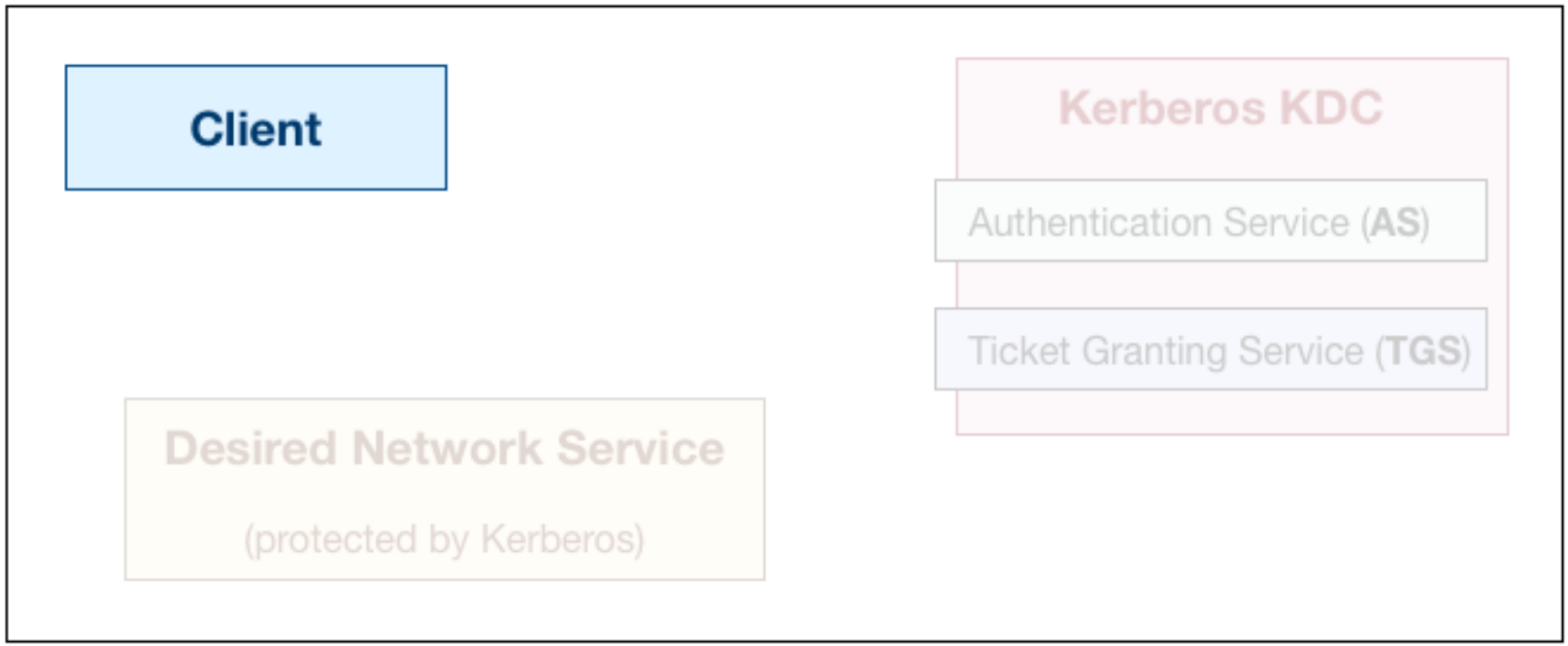
Securing the Kafka Cluster

- Encryption
- **Authentication**
- Authorization
- Auditing
- Essential Points

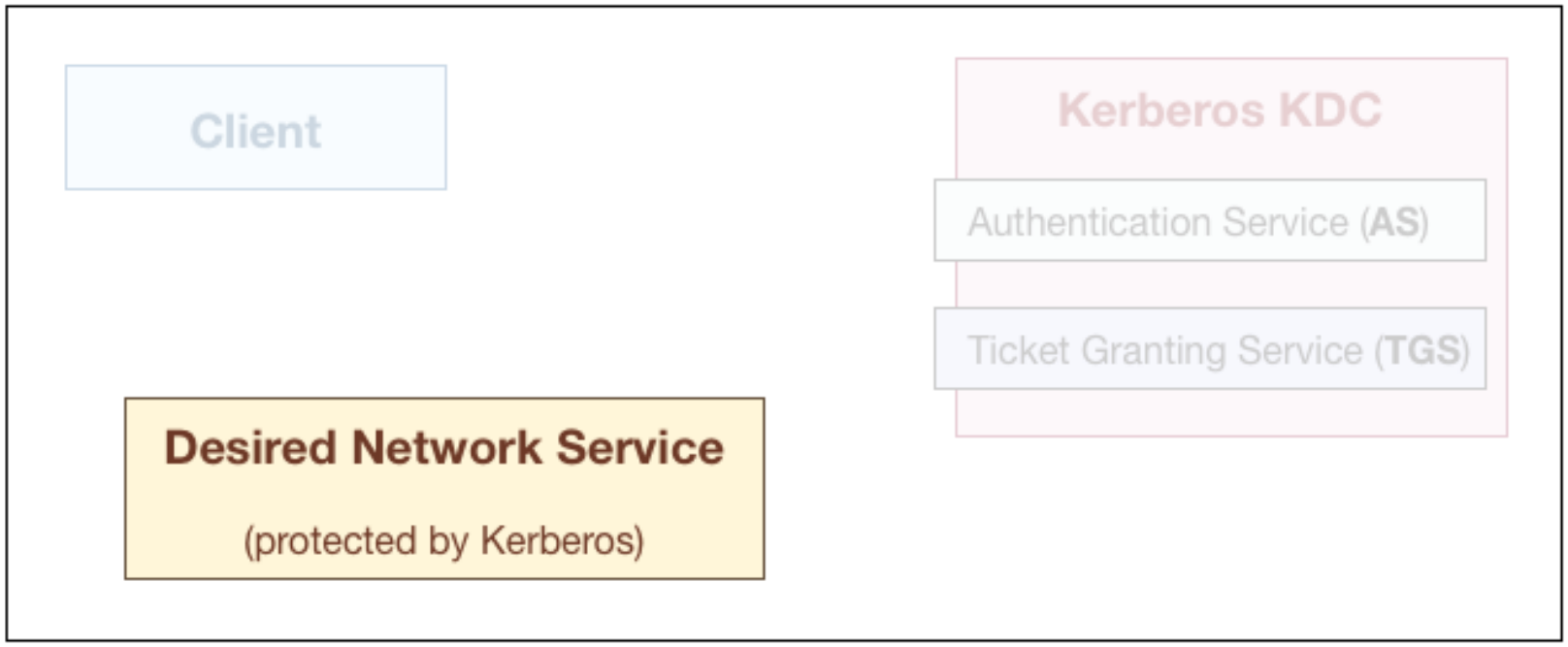
What Is Kerberos?

- **Kerberos is a mature protocol for network authentication**
  - Started at MIT in 1980s
  - Widely used in large UNIX networks in the 1990s
  - Part of Microsoft Active Directory
  - Included in Linux distributions
  - Standard for strong authentication in CDH

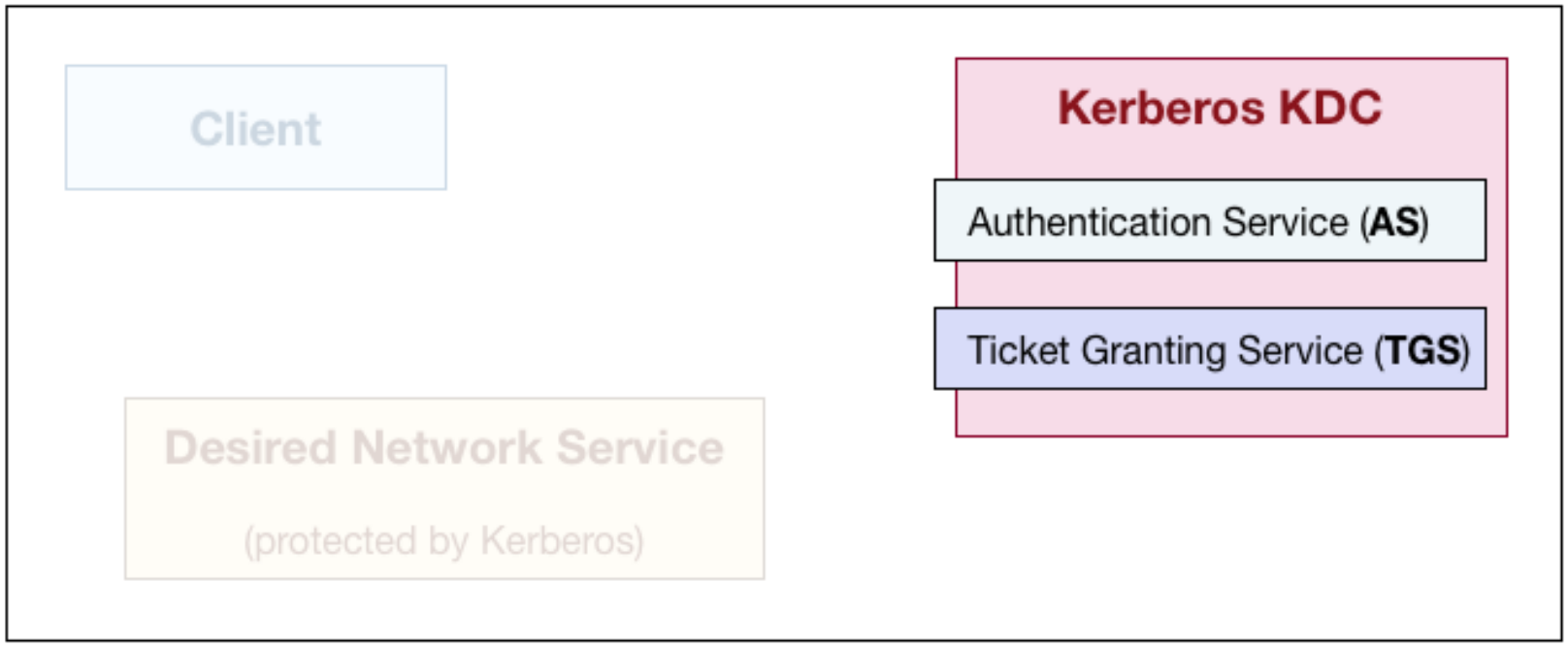
Kerberos Exchange Participants: Client



Kerberos Exchange Participants: Service



Kerberos Exchange Participants: KDC



Realms and Principals

- **A *realm* groups all entities that a KDC may authenticate**
  - Similar in concept to a *domain*
  - By convention, the uppercase version of the DNS zone is used as the realm name
- **A *principal* is a unique identity that can be authenticated**
  - The realm contains a principal for each user (UPN) and service (SPN)

Kerberos Keytabs

- **Short for “key table”**
- **File containing a collection of principals and their keys**
  - Allows one to authenticate as a principal
  - Used when typing a password interactively is impractical
  - Cloudera Manager automatically generates them for services
  - Ensure they’re protected, both in motion and at rest

Kerberos Client Command: `kinit`

- **Initiates authentication sequence, retrieves a TGT, and caches it**
- **Can be used interactively or with a keytab file**
  - Authentication through interactive password challenge

```
$ kinit mdavis@EXAMPLE.COM
Password for kinit mdavis@EXAMPLE.COM:
```

  - Authentication using a keytab

```
$ kinit -kt /home/mdavis/mdavis.keytab mdavis@EXAMPLE.COM
```

Kerberos Client Command: `klist`

- **Displays Kerberos principal and information about cached tickets**
- **Helpful for checking whether you need to run `kinit`**

```
$ klist
Default principal: mdavis@EXAMPLE.COM

Valid starting       Expires              Service principal
01/10/2019 17:18:15  01/10/2019 03:18:15  krbtgt/kdc.example.com@EXAMPLE.COM
 renew until 01/17/2019 17:18:15
```

Kerberos Client Command: `kdestroy`

- **Destroys cached tickets**
- **Not required, but protects cached tickets from attacker**
  - Run `kdestroy` at end of session

```
$ kdestroy
```

Kafka Kerberos Configuration Overview

- **Protocols: `SASL_PLAINTEXT` (SSL disabled, port 9092) or `SASL_SSL` (SSL enabled, port 9093)**
- **Clients specify configuration via properties (protocol, keystore, truststore, etc.)**
- **Clients must also specify JAAS configuration file location**

Kafka SSL-Enabled Kerberized Configuration Properties Example

```
bootstrap.servers=worker1.example.com:9093,worker2.example.com:9093,worke
security.protocol=SASL_SSL
sasl.kerberos.service.name=kafka
ssl.truststore.location=/opt/cloudera/security/pki/
truststore.jks
ssl.truststore.password=mysecret123
ssl.truststore.type=JKS
ssl.keystore.type=JKS
ssl.keystore.location=/opt/cloudera/security/pki/keystore.jks
ssl.keystore.password=mysecret123
ssl.key.password=mysecret123
ssl.enabled.protocols=TLSv1.2,TLSv1.1,TLSv1
```

Client JAAS Configuration File Example

- For command-line clients, location is typically specified via environment variable
- `export KAFKA_OPTS="-Djava.security.auth.login.config=/home/training/jaas.conf"`
- Jn Java code, location is set via `java.security.auth.login.config` system property

```
KafkaClient {
 com.sun.security.auth.module.Krb5LoginModule required
 useTicketCache=true;
};
```

Chapter Topics

Securing the Kafka Cluster

- Encryption
- Authentication
- **Authorization**
- Auditing
- Essential Points

What Is Authorization?

- **Controlling access to a resource**
- **Depends on authentication**
- **Often implemented through *role-based access control***

Resources and Privileges

- **Roles are assigned to groups of users**
- **Roles define *privileges* granted to a *resource***
- ***Resources* are the objects being protected, such as**
  - `table`
  - `collection`
  - `topic`
- ***Privileges* are operations that can be performed on a resource, such as**
  - `read`
  - `write`
  - `create`
  - `delete`

Apache Sentry

- **Provides fine-grained role-based access control for multiple applications**
  - Apache Hive
  - Apache Impala
  - Apache Solr
  - Apache Kafka
- **Relies on underlying authentication systems**
  - On secured clusters, user is a Kerberos principal
  - Relies on system configuration to determine group membership
  - Sentry roles are assigned to groups
- **Use `kafka-sentry` on Sentry gateway to manage roles and privileges**
- **Sentry currently does not protect topic management (ZooKeeper)**

Chapter Topics

Securing the Kafka Cluster

- Encryption
- Authentication
- Authorization
- **Auditing**
- Essential Points

Chapter Topics

Securing the Kafka Cluster

- Encryption
- Authentication
- Authorization
- Auditing
- **Essential Points**

Essential Points

- **In Kafka, you can choose only which parts of security you need**
- **Encryption protects confidentiality of data**
  - Use Cloudera Navigator Encrypt to help protect data at rest
  - Use SSL (TLS) to help protect data in motion
    - Often disabled for inter-broker traffic due to performance penalty
- **Authentication establishes identity**
  - Kerberos provides for strong authentication of users and services
- **Authorization determines whether or not one can perform an operation**
  - Apache Sentry has support for controlling access to Kafka

Bibliography

The following offer more information on topics discussed in this chapter

- **Cloudera Documentation: Configuring Apache Kafka Security**
  - http://tiny.cloudera.com/kafch13a
- **Apache Sentry web site**
  - http://tiny.cloudera.com/kafch13b
- **Cloudera Navigator Encrypt Overview**
  - http://tiny.cloudera.com/kafch13c
- **Cloudera Security Training class (OnDemand)**
  - http://tiny.cloudera.com/kafch13d

cloudera®

Replicating Kafka to Another Cluster

Chapter 14

cloudera®

Course Chapters

- Introduction
- Kafka Overview
- Deploying Apache Kafka
- Kafka Command Line Basics
- Kafka Java API Basics
- Improving Availability through Replication
- Improving Application Reliability
- Improving Application Scalability
- Monitoring and Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka Cluster
- **Replicating Kafka to Another Cluster**
- Conclusion

Chapter Topics

Replicating Kafka to Another Cluster

- **Overview of MirrorMaker**
- Disaster Recovery
- Data Aggregation
- Essential Points

MirrorMaker

- **Overview of Mirror Maker / Replication**
  - Mirror Maker

Chapter Topics

Replicating Kafka to Another Cluster

- Overview of MirrorMaker
- **Disaster Recovery**
- Data Aggregation
- Essential Points

Chapter Topics

Replicating Kafka to Another Cluster

- Overview of MirrorMaker
- Disaster Recovery
- **Data Aggregation**
- Essential Points

Chapter Topics

Replicating Kafka to Another Cluster

- Overview of MirrorMaker
- Disaster Recovery
- Data Aggregation
- **Essential Points**

cloudera®

Conclusion

Chapter 15

cloudera®

Course Chapters

- Introduction
- Kafka Overview
- Deploying Apache Kafka
- Kafka Command Line Basics
- Kafka Java API Basics
- Improving Availability through Replication
- Improving Application Reliability
- Improving Application Scalability
- Monitoring and Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka Cluster
- Replicating Kafka to Another Cluster
- **Conclusion**

Course Objectives (1)

During this course, you have learned

- **What Apache Kafka is and how organizations are using it**
- **Which roles producers, consumers, and brokers play in Kafka**
- **How Cloudera's distribution of Kafka relates to Apache Kafka**
- **What to consider when planning a Kafka installation**
- **How to deploy a typical Kafka cluster using Cloudera Manager**
- **How to manage topics from both the command line and custom Java code**
- **How to produce and consume messages from the command line and custom Java code**
- **How to increase fault tolerance through replication**

Course Objectives (2)

- How several configuration properties affect message delivery and storage
- How messages are partitioned for scalability and how it affects clients
- Which metrics are most important to monitor
- How to troubleshoot common problems and performance issues
- How to improve efficiency by using compression and custom serialization
- Which technologies Cloudera recommends for securing Kafka
- How you can mirror Kafka data to another cluster

Which Course to Take Next

- **For developers**
  - *Developer Training for Spark and Hadoop*
  - *Cloudera Search Training*
  - *Cloudera Training for Apache HBase*
- **For system administrators**
  - *Cloudera Administrator Training for Apache Hadoop*
  - *Cloudera Security Training*
- **For data analysts and data scientists**
  - *Cloudera Data Analyst Training*
  - *Cloudera Data Scientist Training*