

**CLOUDERA**

Educational Services

# Cloudera Training for Apache Kafka



## Introduction

---

Chapter 1

# Course Chapters

---

- **Introduction**
- Kafka Overview
- Deploying Apache Kafka
- Kafka Command Line Basics
- Using Streams Messaging Manager (SMM)
- Kafka Java API Basics
- Improving Availability through Replication
- Improving Application Scalability
- Improving Application Reliability
- Analyzing Kafka Clusters with SMM
- Monitoring Kafka
- Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka 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 Hive	Apache Pig
Apache Avro	Apache Impala	Apache Ranger
Apache Ambari	Apache Kafka	Apache Sentry
Apache Atlas	Apache Knox	Apache Solr
Apache Bigtop	Apache Kudu	Apache Spark
Apache Crunch	Apache Lucene	Apache Sqoop
Apache Druid	Apache Mahout	Apache Storm
Apache Flink	Apache NiFi	Apache Tez
Apache Flume	Apache Oozie	Apache Tika
Apache Hadoop	Apache ORC	Apache Zeppelin
Apache HBase	Apache Parquet	Apache ZooKeeper
Apache HCatalog	Apache Phoenix	

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

# Chapter Topics

---

## Introduction

- **About This Course**
- Introductions
- About Cloudera
- About Cloudera Educational Services
- Course Logistics
- 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
- Understand fundamental resource considerations for planning Kafka deployments

# Chapter Topics

---

## Introduction

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

# Introductions

---

- **About your instructor**
- **About you**
  - Currently, what do you do at your workplace?
  - What is your experience with database technologies, programming, and query languages?
  - How much experience do you have with UNIX or Linux?
  - What is your experience with big data?
  - What do you expect to gain from this course? What would you like to be able to do at the end that you cannot do now?

# Chapter Topics

---

## Introduction

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

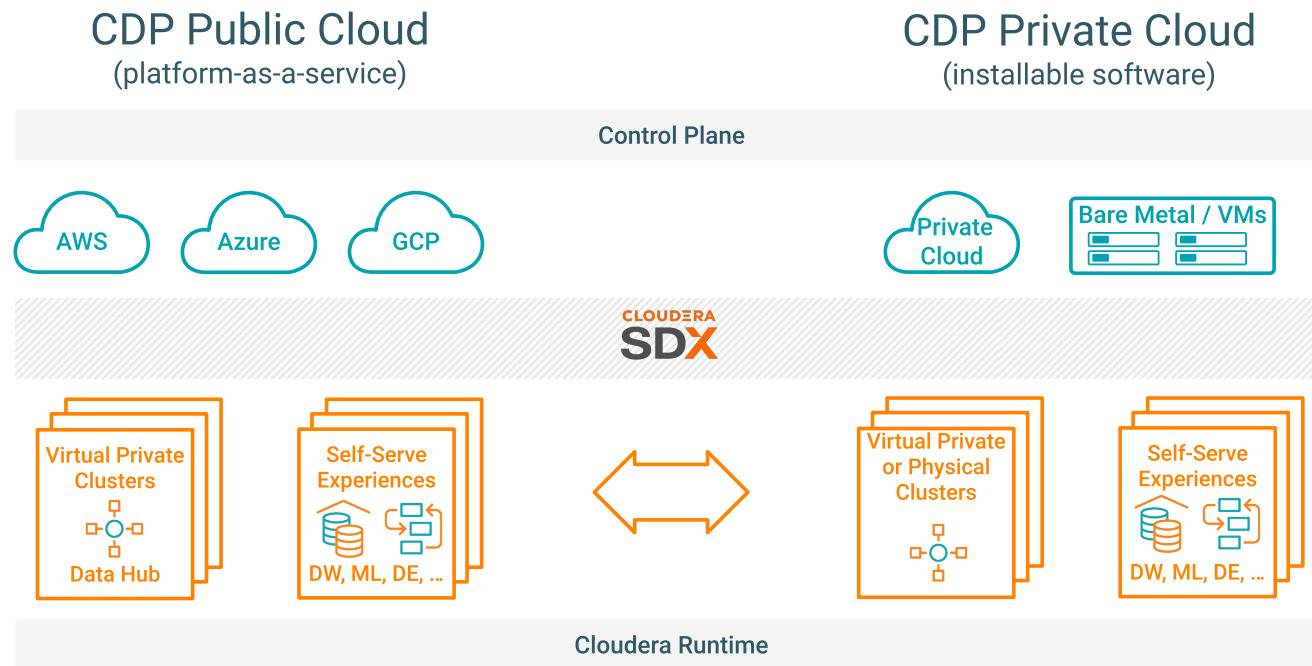
## About Cloudera

---



- Cloudera (founded 2008) and Hortonworks (founded 2011) merged in 2019
- The new Cloudera improves on the best of both companies
  - Introduced the world's first Enterprise Data Cloud
  - Delivers a comprehensive platform for any data from the Edge to AI
  - Leads in training, certification, support, and consulting for data professionals
  - Remains committed to open source and open standards

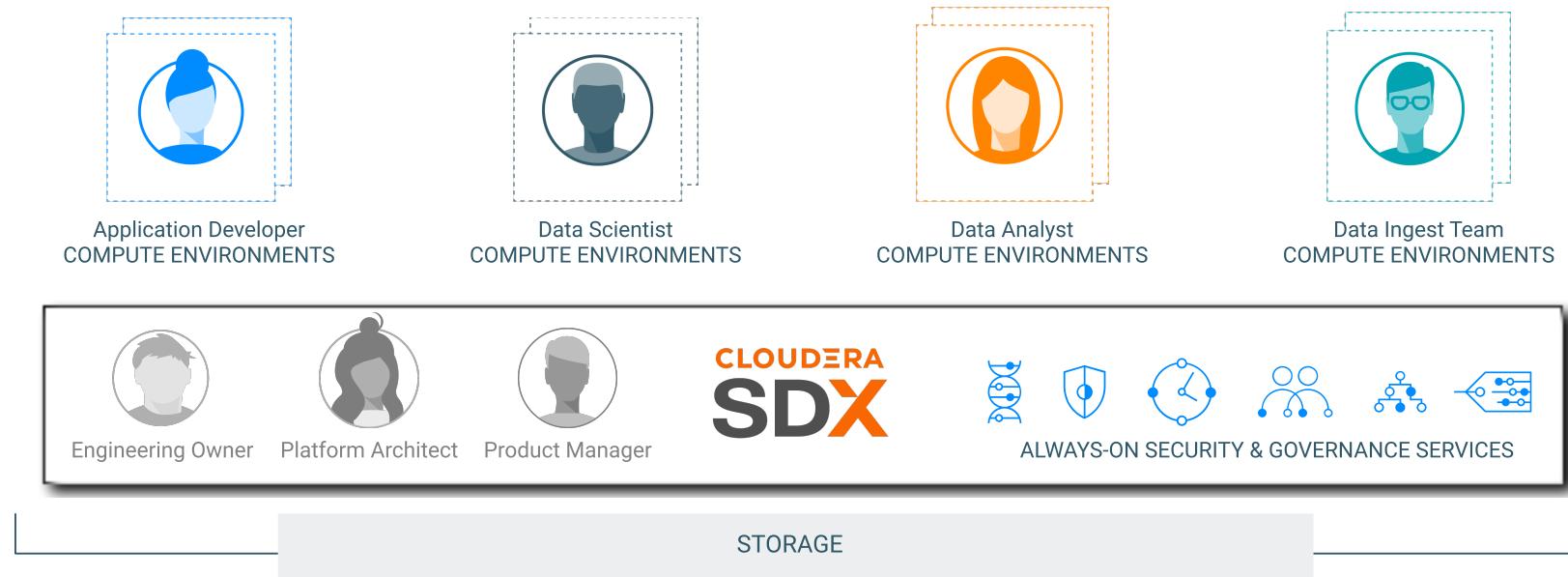
# Cloudera Data Platform



A suite of products to collect, curate, report, serve, and predict

- **Cloud native or bare metal deployment**
- **Powered by open source**
- **Analytics from the Edge to AI**
- **Unified data control plane**
- **Shared Data Experience (SDX)**

# Cloudera Shared Data Experience (SDX)



- **Full data lifecycle:** Manages your data from ingestion to actionable insights
- **Unified security:** Protects sensitive data with consistent controls
- **Consistent governance:** Enables safe self-service access

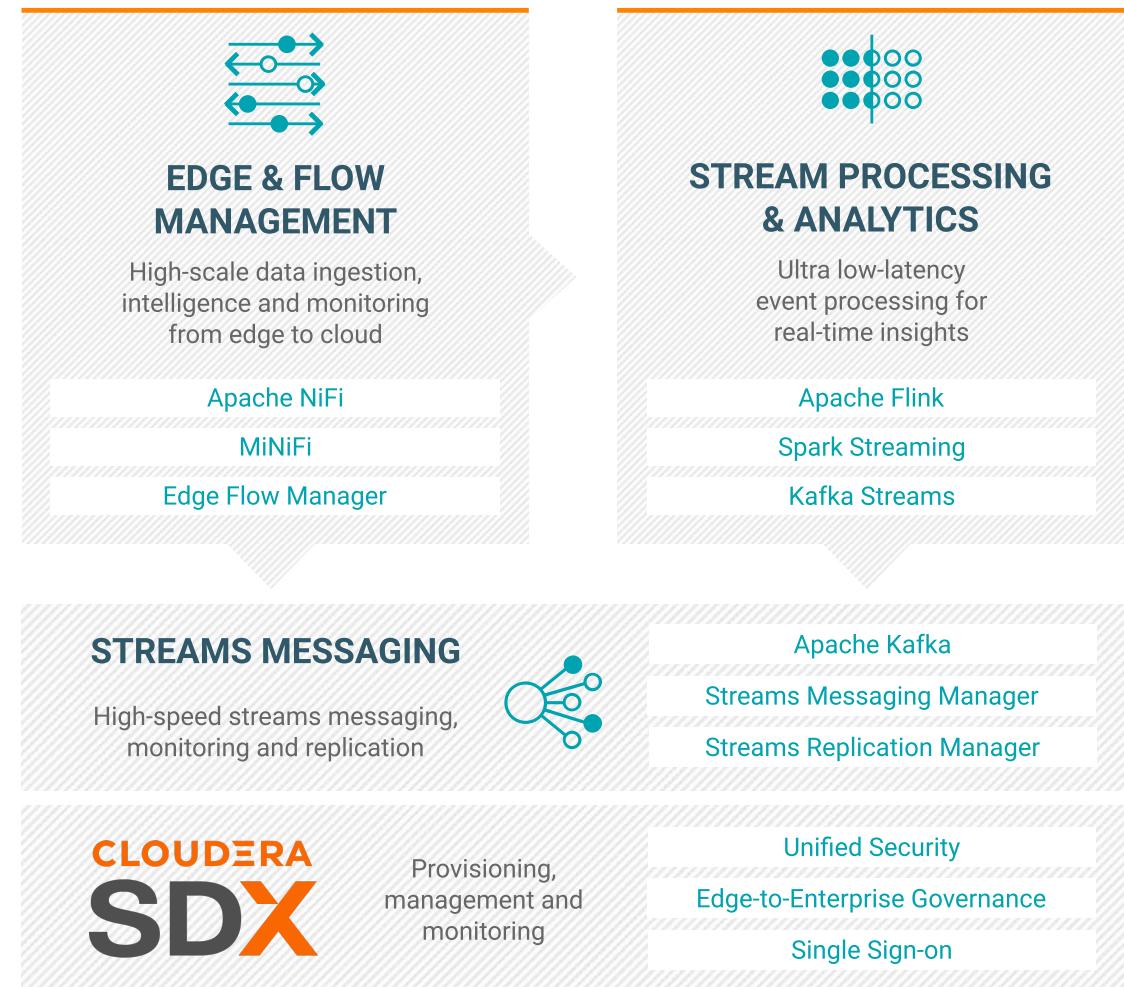
# Self-Serve Experiences for Cloud Form Factors

- Services customized for specific steps in the data lifecycle
  - Emphasize productivity and ease of use
  - Auto-scale compute resources to match changing demands
  - Isolate compute resources to maintain workload performance

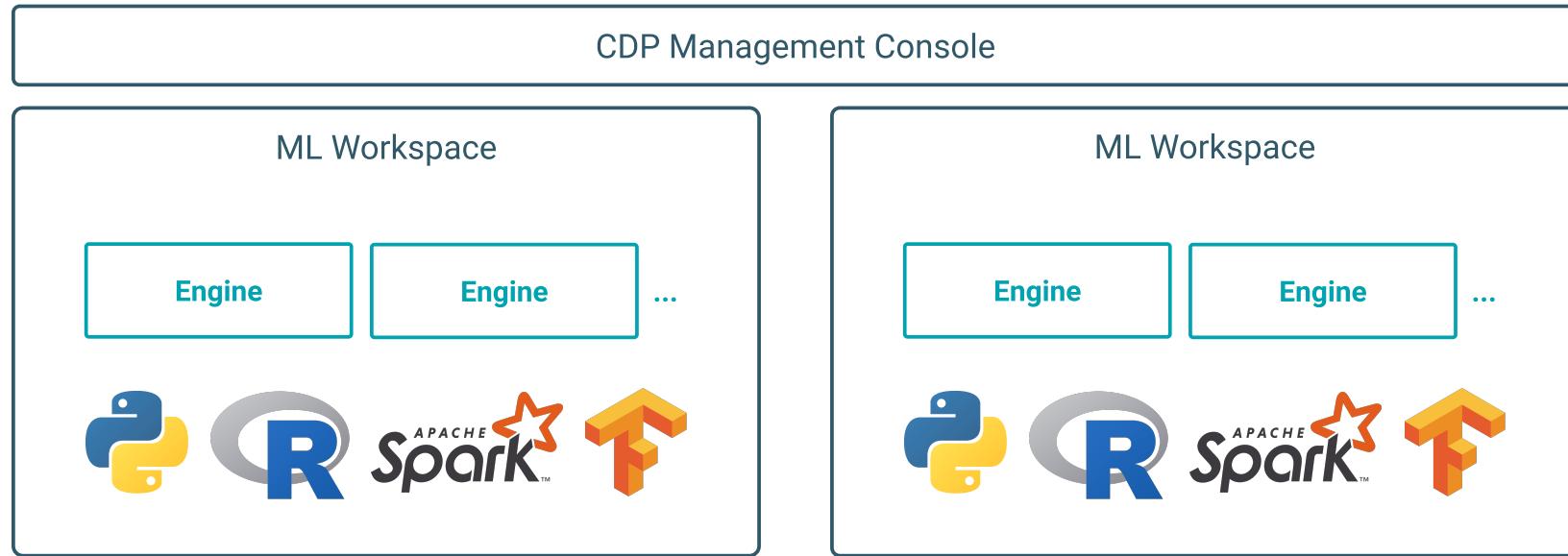


# Cloudera DataFlow

- Data-in-motion platform
- Reduces data integration development time
- Manages and secures your data from edge to enterprise

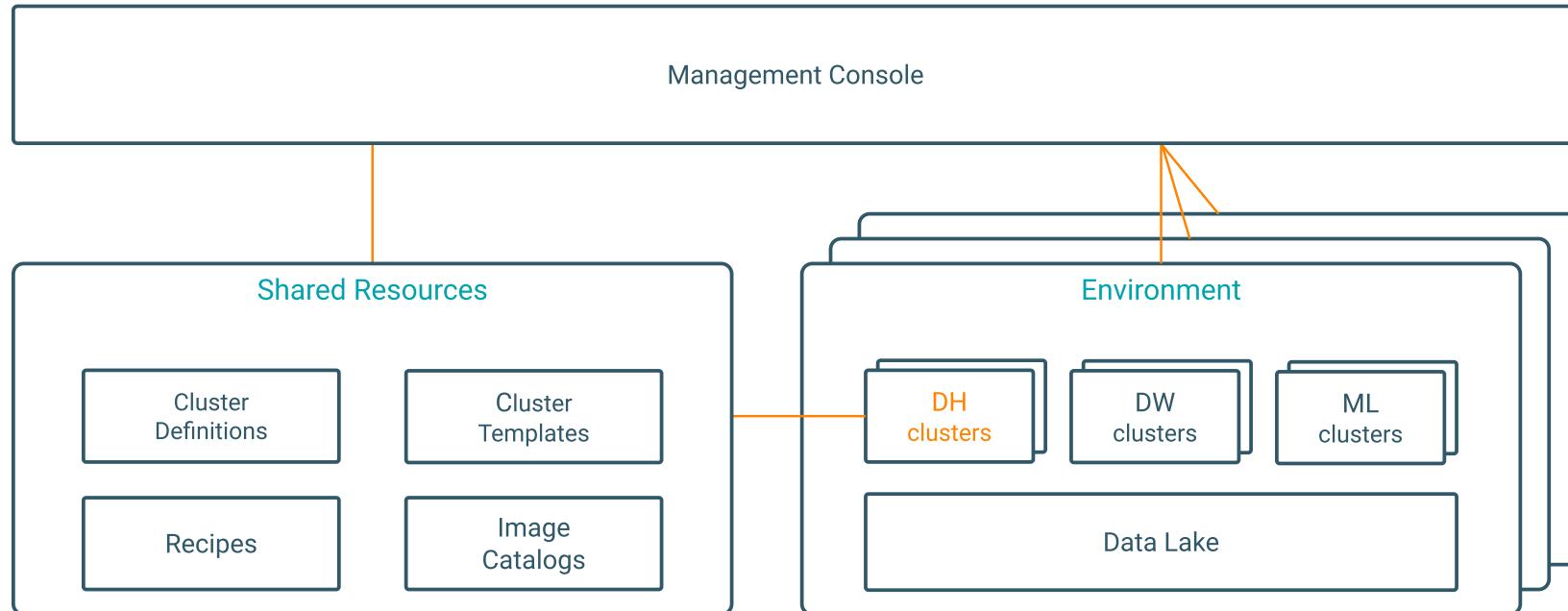


# Cloudera Machine Learning



- **Cloud-native enterprise machine learning**
  - Fast, easy, and secure self-service data science in enterprise environments
  - Direct access to a secure cluster running Spark and other tools
  - Isolated environments for running Python, R, and Scala code
  - Teams, version control, collaboration, and project sharing

# Cloudera Data Hub



## Customize your own experience in cloud form factors

- Integrated suite of analytic engines
- Cloudera SDX applies consistent security and governance
- Fueled by open source innovation

# Chapter Topics

---

## Introduction

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

# Cloudera Educational Services

---

- **We offer a variety of ways to take our courses**
  - Instructor-led, both in physical and virtual classrooms
  - Private and customized courses also available
  - Self-paced, through Cloudera OnDemand
- **Courses for all kinds of data professionals**
  - Executives and managers
  - Data scientists and machine learning specialists
  - Data analysts
  - Developers and data engineers
  - System administrators
  - Security professionals

# Cloudera Education Catalog

- A broad portfolio across multiple platforms
  - Not all courses shown here
  - See [our website](#) for the complete catalog

	Administrator	Security	NiFi	AWS Fundamentals for CDP	
ADMINISTRATOR	CDH   HDP   CDP	CDH   HDP	CDF		
DATA ANALYST	Data Analyst CDH   CDP	Hive 3 HDP	Kudu CDH	Cloudera Data Warehouse CDP	
DEVELOPER & DATA ENGINEER	Spark CDH   HDP	Spark Performance Tuning CDH	Stream Developer CDF	Kafka CDP	Search   Solr CDH
DATA SCIENTIST	Data Scientist CDH   HDP   CDP	Cloudera DS Workbench CDH   HDP		CML CDP	Architecture Workshop CDH

# Cloudera OnDemand

---

- Our OnDemand catalog includes
  - Courses for developers, data analysts, administrators, and data scientists, updated regularly
  - Exclusive OnDemand-only courses, such as those covering security and Cloudera Data Science Workbench
  - Free courses such as *Essentials* and *Cloudera Director* available to all with or without an OnDemand account
- Features include
  - Video lectures and demonstrations with searchable transcripts
  - Hands-on exercises through a browser-based virtual environment
  - Discussion forums monitored by Cloudera course instructors
  - Searchable content within and across courses
- 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](#)

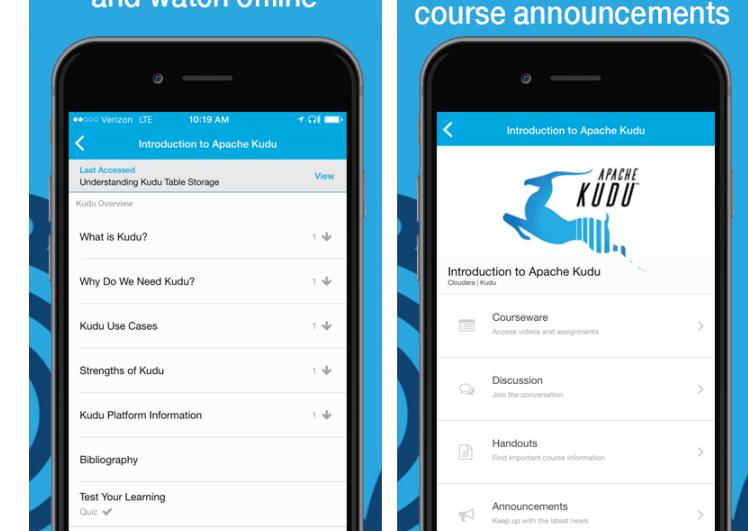
# Accessing Cloudera OnDemand

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

The screenshot shows a web browser window with the following details:

- Header:** Home, Course (selected), Discussion, Progress.
- Search Bar:** Search course content...
- Sidebar:** Introduction to Apache Hadoop and the Hadoop Ecosystem, Apache Hadoop Overview (selected), Data Storage and Ingest, Data Processing, Data Analysis and Exploration, Other Ecosystem Tools, Intro to the Hands-On Exercises, Hands-On Exercise: Accessing the Exercise Environment, Hands-On Exercise: General Notes, Hands-On Exercise: Query Hadoop Data with Apache Impala, Test Your Learning, Quiz.
- Main Content:** Introduction to Apache Hadoop and the Hadoop Ecosystem > Apache Hadoop Overview > Apache Hadoop Overview. The title is "Apache Hadoop Overview".
  - Section:** Apache Hadoop Overview
  - Content:** Common Hadoop Use Cases
    - Extract, Transform, and Load (ETL)
    - Data analysis
    - Text mining
    - Index building
    - Graph creation and analysis
    - Pattern recognition
    - Data storage
    - Collaborative filtering
    - Prediction models
    - Sentiment analysis
    - Risk assessment
  - Text:** What do these workloads have in common? Nature of the data...
    - Volume
    - Velocity
    - Variety
- Bottom Right:** Start of transcript. Skip to the end.

- Cloudera OnDemand is also available through an iOS app
  - Search for “Cloudera OnDemand” in the iOS App Store



# Cloudera Certification

---

- The leader in Apache Hadoop-based certification
- Cloudera certification exams favor hands-on, performance-based problems that require execution of a set of real-world tasks against a live, working cluster
- We offer two levels of certifications
  - Cloudera Certified Associate (CCA)
    - CCA Spark and Hadoop Developer
    - CCA Data Analyst
    - CCA CDH Administrator and CCA HDP Administrator
  - Cloudera Certified Professional (CCP)
    - CCP Data Engineer

# Chapter Topics

---

## Introduction

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

# Logistics

---

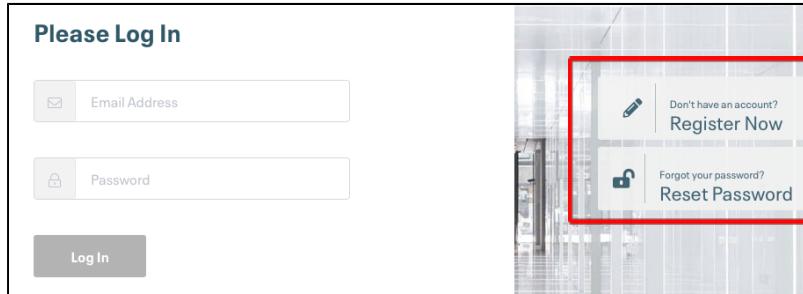
- Class start and finish time
- Lunch
- Breaks
- Restrooms
- Wi-Fi access
- Virtual machines

# Downloading the Course Materials

---

## 1. Log in using <https://university.cloudera.com/user>

- If necessary, use the **Register Now** link on the right to create an account
- If you have forgotten your password, use the **Reset Password** link



## 2. Scroll down to find this course

- If necessary, click **My Learning** under the photo
- You may also want to use the **Current** filter

## 3. Select the course title

## 4. Click the **Resources** tab

## 5. Click a file to download it

# Chapter Topics

---

## Introduction

- About This Course
- Introductions
- About Cloudera
- About Cloudera Educational Services
- Course Logistics
- **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 six-node cluster**
  - Managed by Cloudera Manager
  - Versions used
    - Cloudera Manager
    - Cloudera Data Platform (CDP) Private Base
    - 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

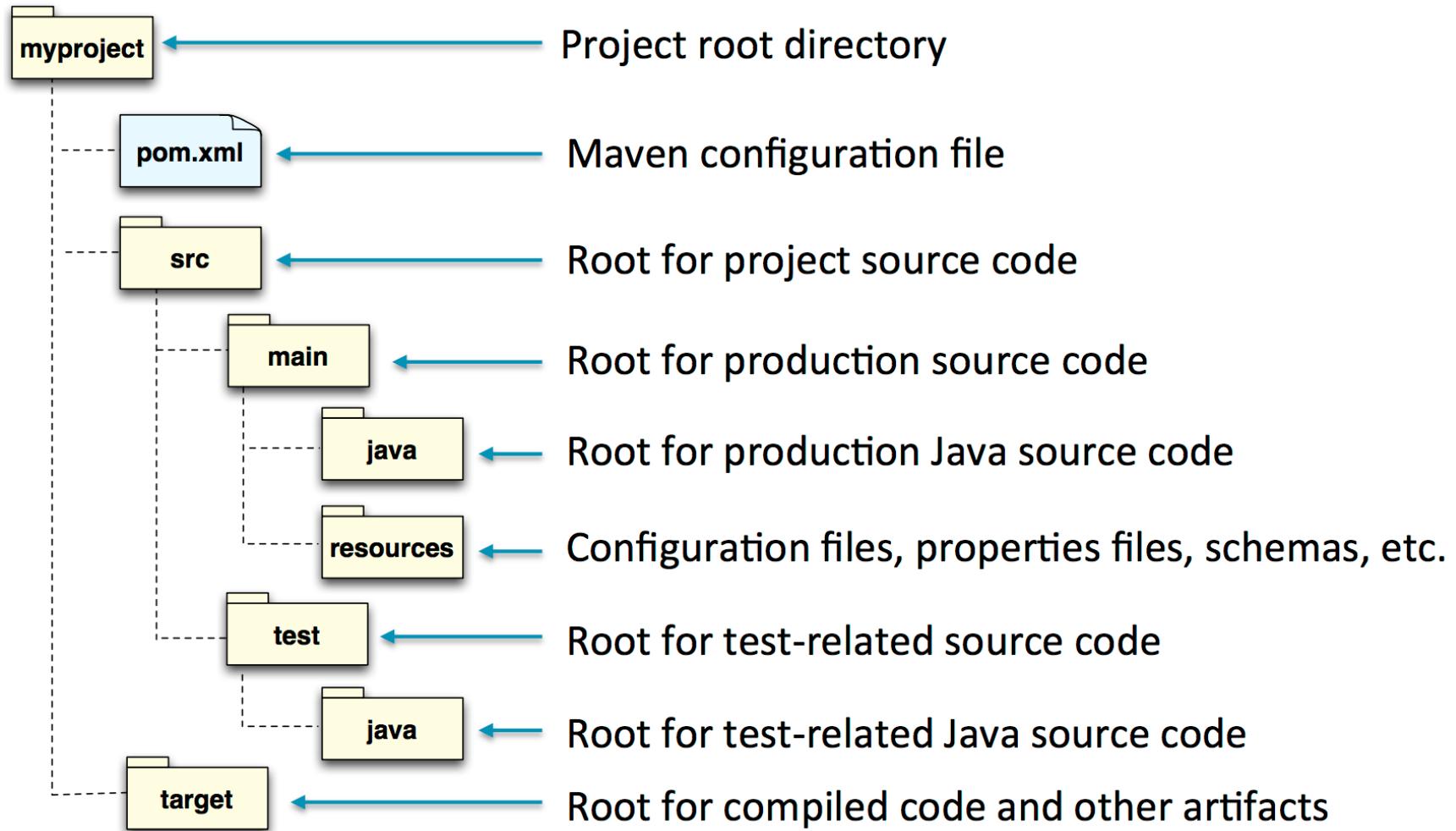
---

- 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"
```



## Kafka Overview

---

Chapter 2

# Course Chapters

---

- Introduction
- **Kafka Overview**
- Deploying Apache Kafka
- Kafka Command Line Basics
- Using Streams Messaging Manager (SMM)
- Kafka Java API Basics
- Improving Availability through Replication
- Improving Application Scalability
- Improving Application Reliability
- Analyzing Kafka Clusters with SMM
- Monitoring Kafka
- Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka Cluster
- Conclusion

# Kafka Overview

---

**By the end of this chapter, you will be able to**

- **Understand how the Kafka distributed platform is used for streaming data**
- **Describe the flow of message streams**
- **Identify common use cases**

# Chapter Topics

---

## Kafka Overview

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

# What Is Apache Kafka?

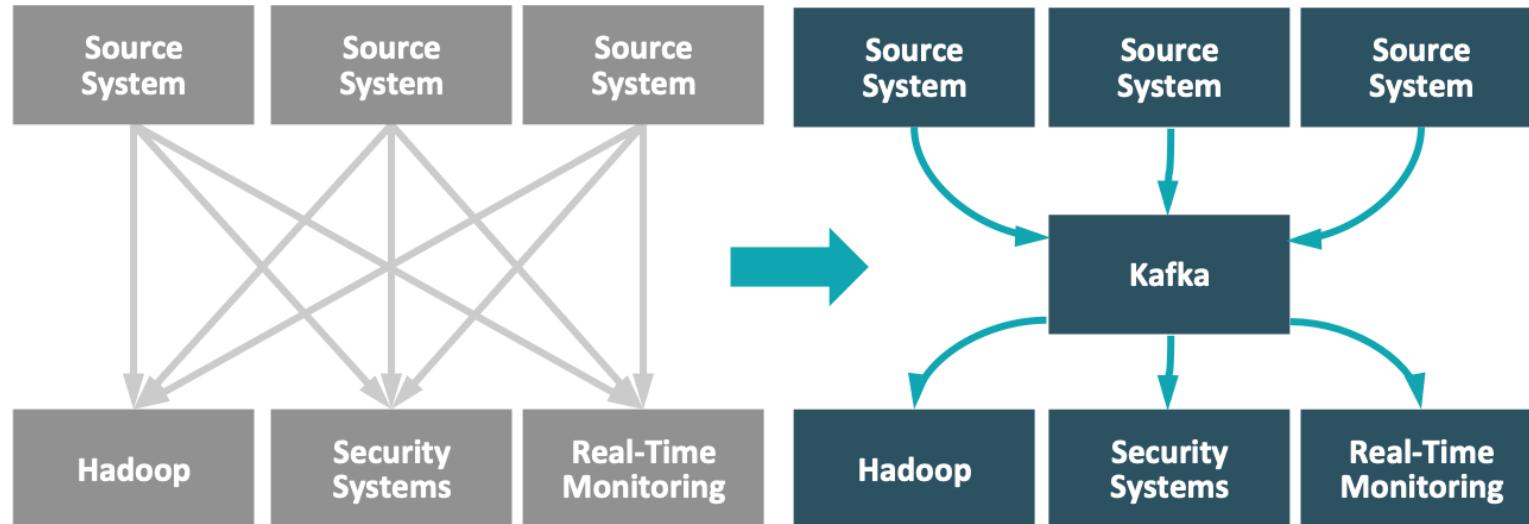
---

- **Apache Kafka is a fast, scalable, distributed publish-subscribe messaging system**
  - Widely used for data ingest
  - Offers scalability, performance, reliability, and flexibility
- **Originally created at LinkedIn, now an open source Apache project**
  - Donated to the Apache Software Foundation in 2011
  - Graduated from the Apache Incubator in 2012



# Kafka Decouples Production and Consumption of Data

- Decouples production and consumption of messages
- Organized by topic to support several use cases



# Why Use Apache Kafka?

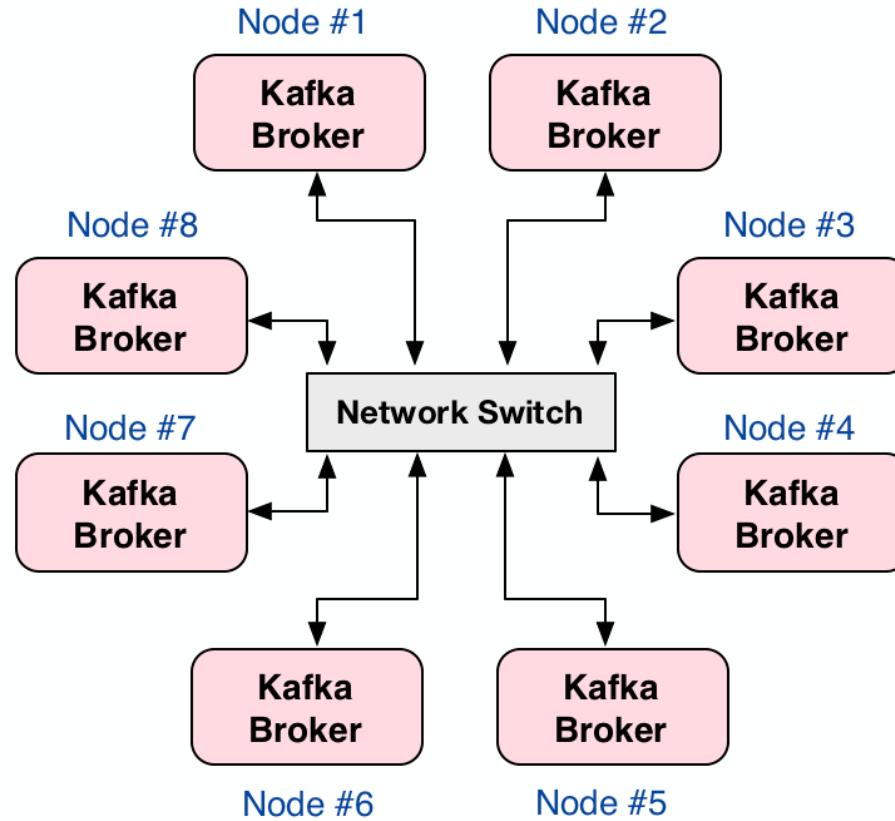
---

- **High throughput, low latency**
- **Durability and reliability**
- **Horizontally scalable**
- **Efficient implementation to operate at speed with big data volumes**

# Kafka Clusters

---

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



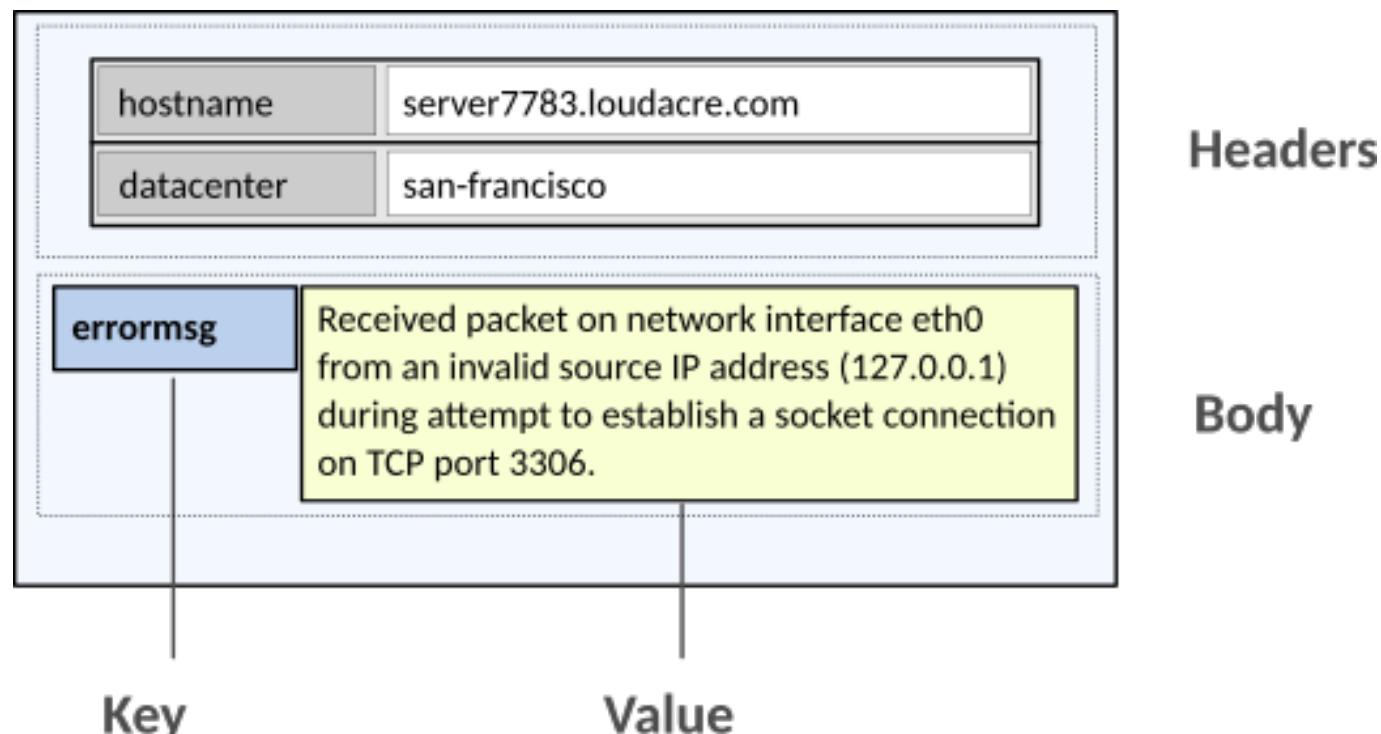
# Kafka Messages

---

- **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

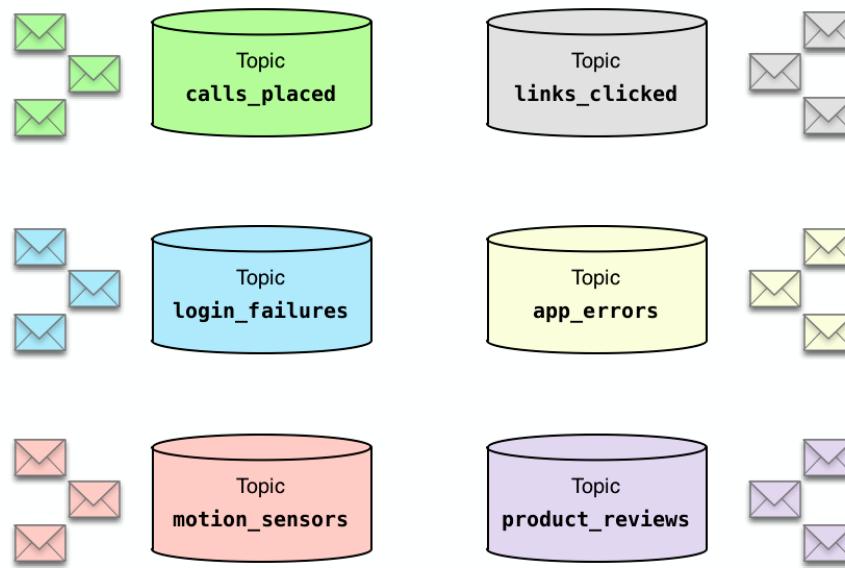
- Headers are an optional set of name-value pairs
  - Typically used for metadata and message processing
  - Name is of type `String`, value is a `byte[]`
- The payload is sent in the record's body
  - Consists of a key (often empty) and value
  - Both can be of any type, but each is ultimately converted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



# Kafka Topics

---

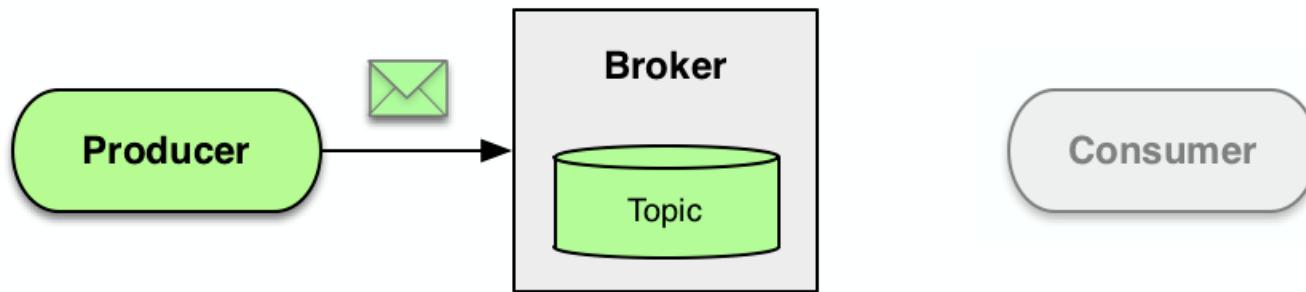
- **Messages are classified by categories, known as *topics***
  - Kafka clients specify the topic when sending and receiving messages
  - Kafka brokers use topics to organize and store data



# Kafka Producers

---

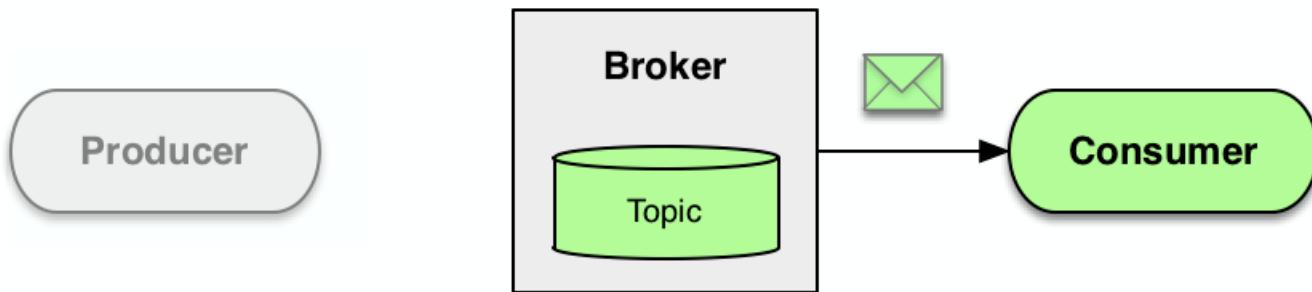
- Producers are Kafka clients that publish messages to a topic
  - Can be configured to retry if sending fails
  - For efficiency, producers can send a batch of messages to a broker



# Kafka Consumers

---

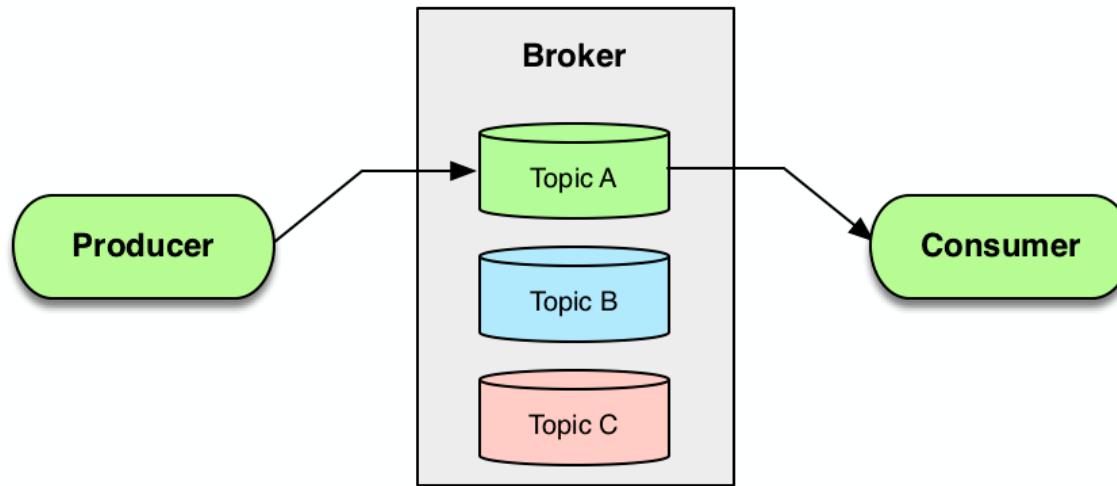
- Consumers are Kafka clients that fetch messages from a topic
  - They poll the broker for messages
  - Producers and consumers are decoupled from one another



# Multiple Topics

---

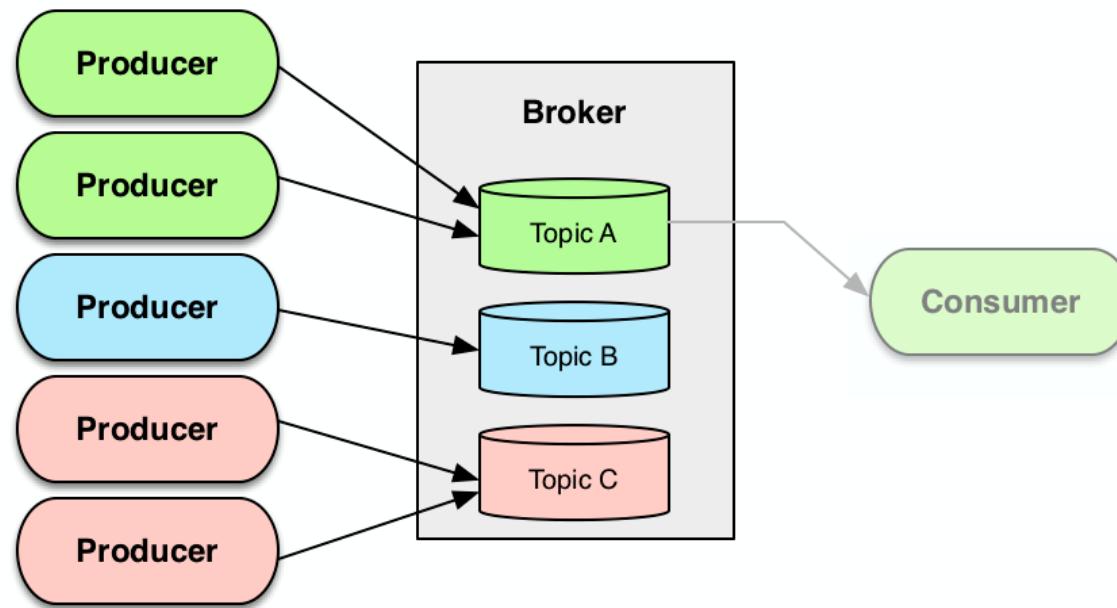
- Each broker may have responsibility for multiple topics



# Multiple Producers

---

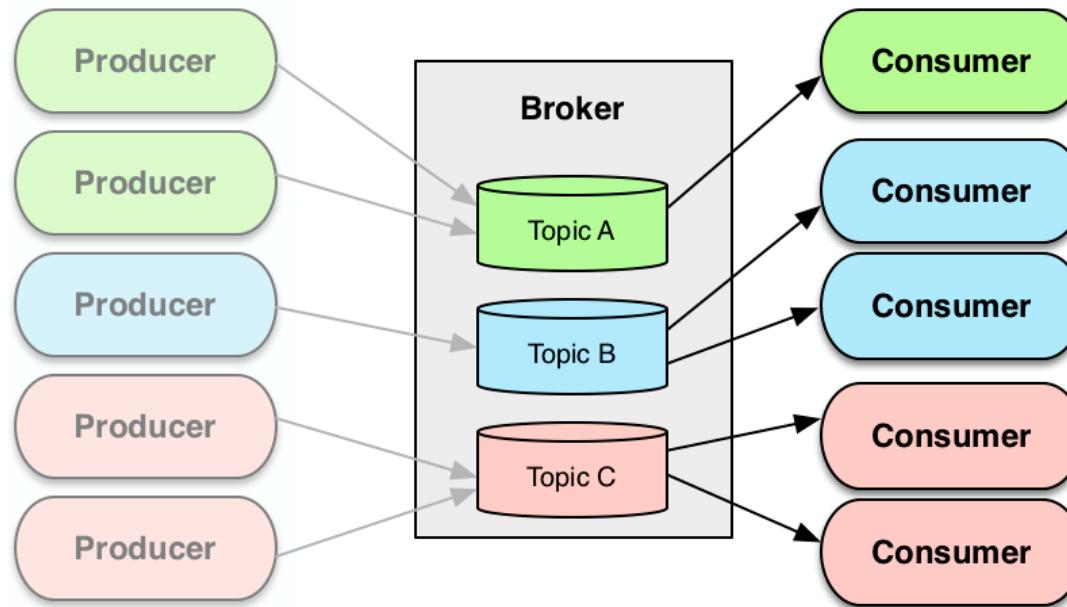
- Each topic may have many producers publishing messages to it



# Multiple Consumers

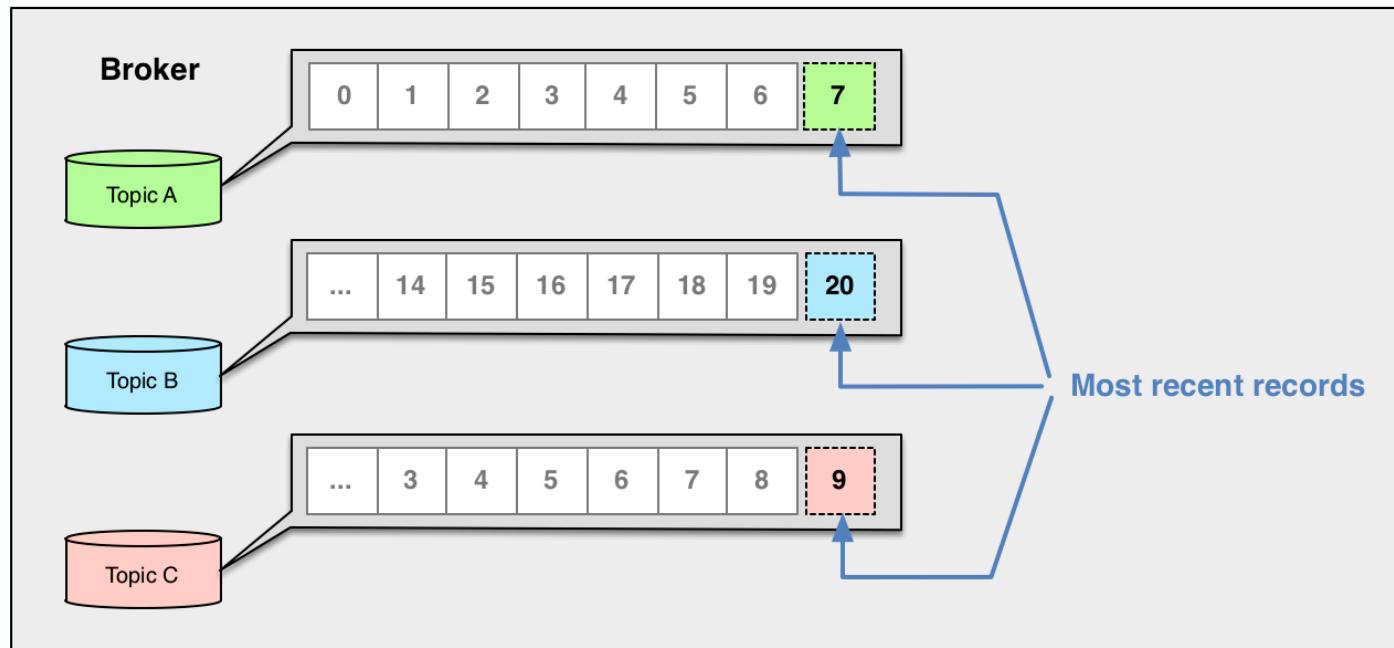
---

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



# Message Storage and Retention

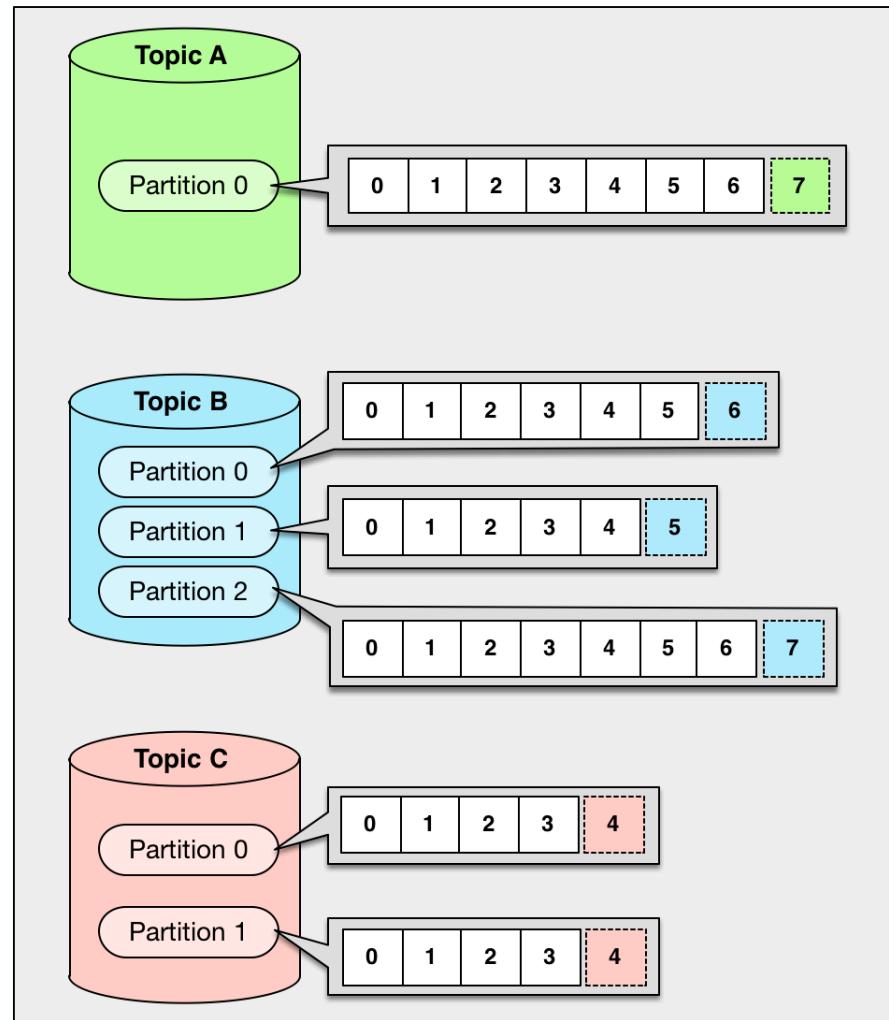
- Messages are immutable and stored in the order sent
  - May be deleted after the specified retention period has elapsed
  - Retention period is configurable on a per-topic basis



# Partitioning

---

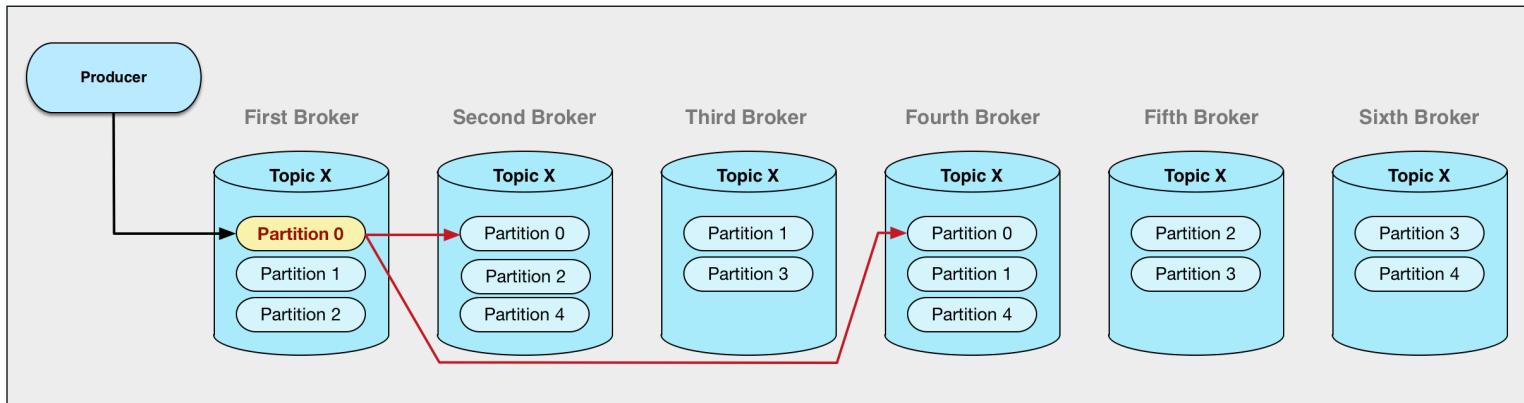
- For better scalability, each topic can be divided into multiple *partitions*



# 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

# CDP Support of Apache Kafka

---

- **Kafka is a CDP component of Streams Messaging**
- **Each CDP release includes a specific version of Apache Kafka**
  - For example, CDP Private Cloud Base 7.1 supports Apache Kafka 2.4
  - Includes backports and bugfixes from upstream project
- **Check CDP release notes for important information**
  - Experimental or unsupported features
  - Known issues and workarounds

# 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](#)
- [Original LinkedIn Blog Article about Kafka](#)
- [Cloudera's Apache Kafka Guide](#)



# Deploying Apache Kafka

---

Chapter 3

# Course Chapters

---

- Introduction
- Kafka Overview
- **Deploying Apache Kafka**
- Kafka Command Line Basics
- Using Streams Messaging Manager (SMM)
- Kafka Java API Basics
- Improving Availability through Replication
- Improving Application Scalability
- Improving Application Reliability
- Analyzing Kafka Clusters with SMM
- Monitoring Kafka
- Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka Cluster
- Conclusion

# Deploying Kafka

---

By the end of this chapter, you will be able to

- Identify system requirements for deploying Kafka
- Use Cloudera Manager to deploy, monitor, and manage services
- Deploy Kafka for maximum performance and availability

# Chapter Topics

---

## Deploying Apache Kafka

- **System Requirements and Dependencies**
- Service Roles
- Planning Your Deployment
- Deploying Kafka Services
- Hands-On Exercise: Preparing the Exercise Environment
- Hands-On Exercise: Installing the Kafka Service with Cloudera Manager
- Optional Hands-On Exercise: Create Metrics Dashboards
- Optional Hands-On Exercise: Using the CM API
- Essential Points

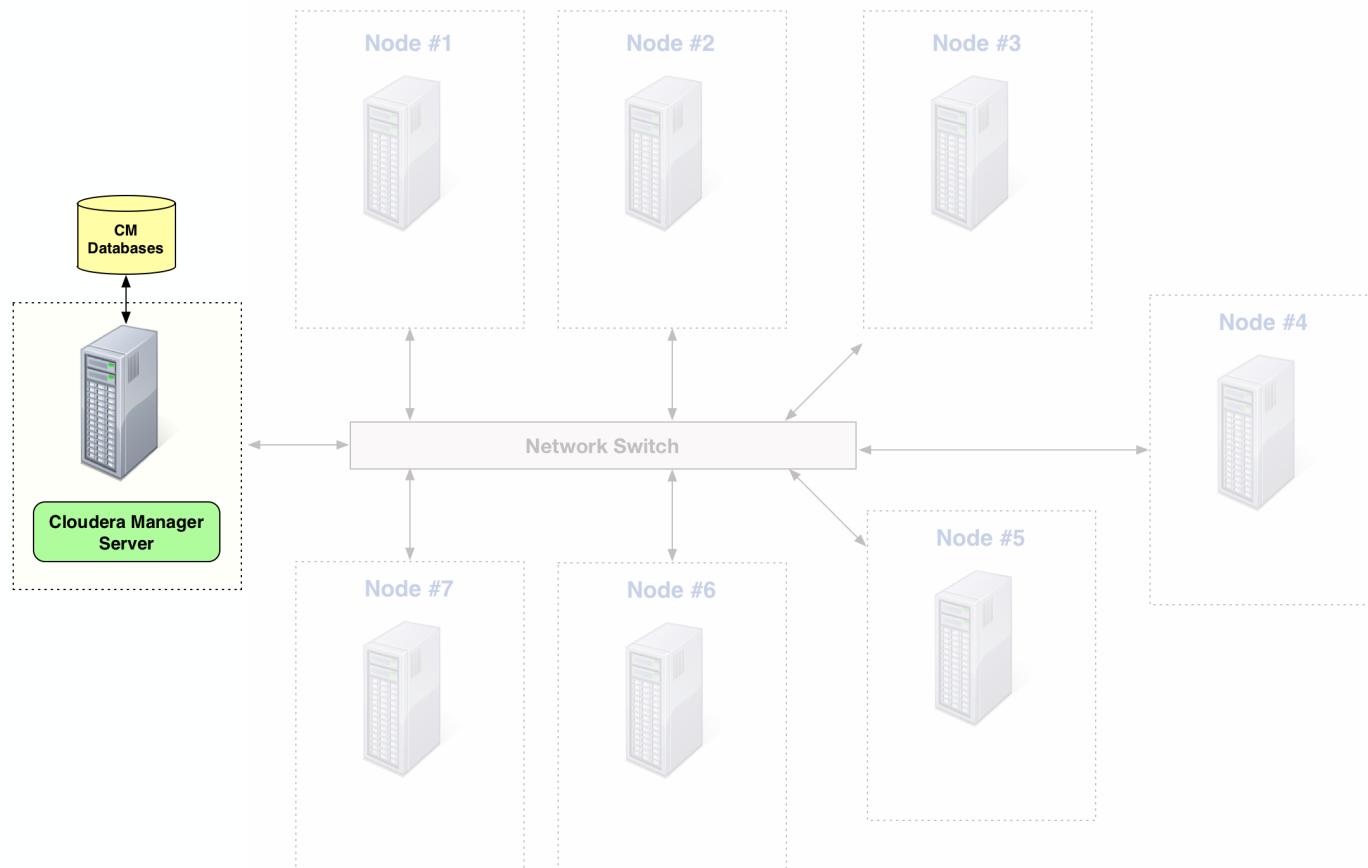
# CDP System Requirements

---

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

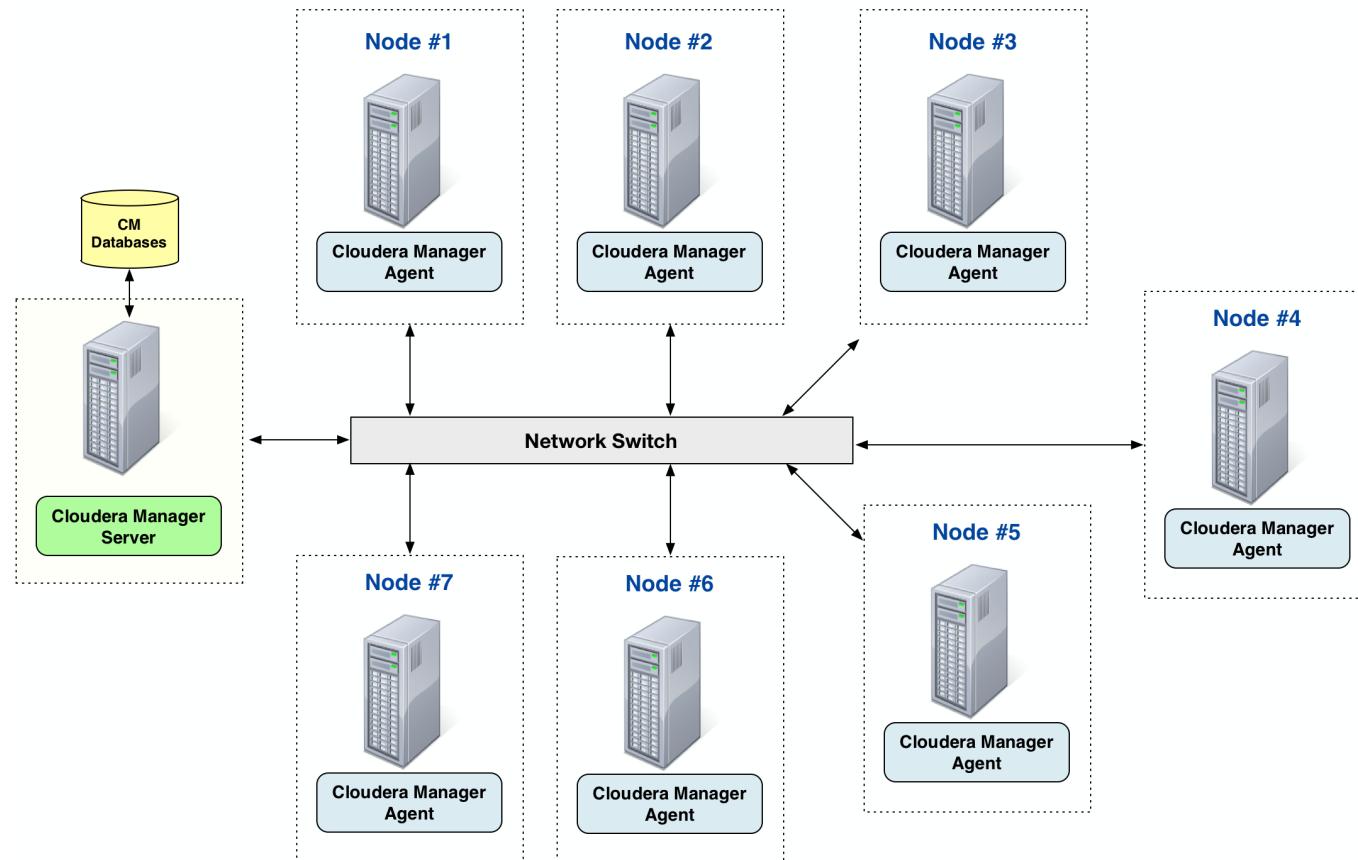
# Cloudera Manager: Server

- 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

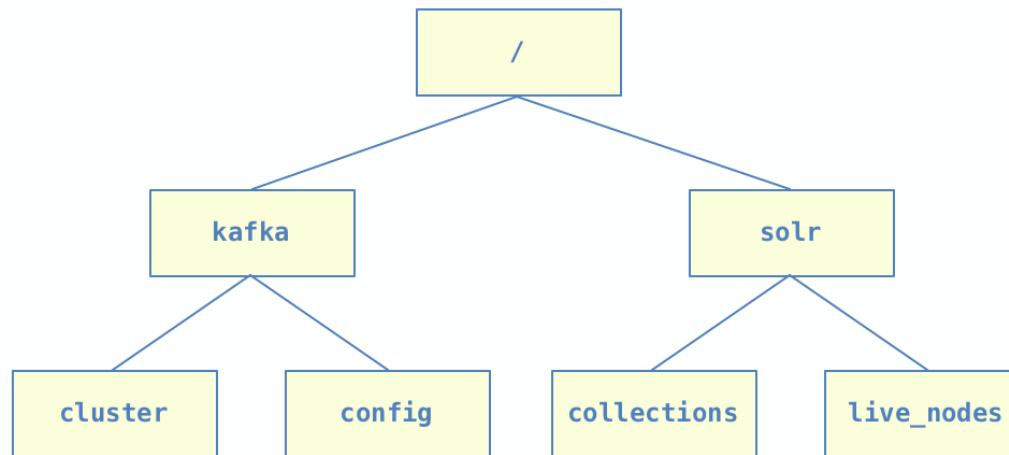
- Cloudera Manager agents run on each node in the cluster
- Agents collect statistics and manage local services on the node
  - They receive commands from, and report status to, the server



# 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



# Chapter Topics

---

## Deploying Apache Kafka

- System Requirements and Dependencies
- **Service Roles**
- Planning Your Deployment
- Deploying Kafka Services
- Hands-On Exercise: Preparing the Exercise Environment
- Hands-On Exercise: Installing the Kafka Service with Cloudera Manager
- Optional Hands-On Exercise: Create Metrics Dashboards
- Optional Hands-On Exercise: Using the CM API
- Essential Points

# Kafka service 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

# Chapter Topics

---

## Deploying Apache Kafka

- System Requirements and Dependencies
- Service Roles
- **Planning Your Deployment**
- Deploying Kafka Services
- Hands-On Exercise: Preparing the Exercise Environment
- Hands-On Exercise: Installing the Kafka Service with Cloudera Manager
- Optional Hands-On Exercise: Create Metrics Dashboards
- Optional Hands-On Exercise: Using the CM API
- Essential Points

# Typical Cluster Hardware

---

- **CDP Kafka Hardware Recommendations**

- Avoid clusters that span multiple datacenters
  - Common to run ZooKeeper (dedicated to Kafka) on 3 broker nodes
  - For large, production Kafka environments it's recommended to use dedicated ZooKeeper hosts

# 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
  - RAID 10 is recommended for Kafka Broker storage disks
- Estimate based on message size, send rates, retention, and replication
  - Plan for future growth
  - Be conservative: More capacity is always better
  - Kafka is known to have issues when storage disks reach capacity
- Tradeoffs
  - 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: Preparing the Exercise Environment
- Hands-On Exercise: Installing the Kafka Service with Cloudera Manager
- Optional Hands-On Exercise: Create Metrics Dashboards
- Optional Hands-On Exercise: Using the CM API
- Essential Points

## Separate Cluster or Shared with other CDP Services?

---

- Kafka can coexist alongside other CDP 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: Preparing the Exercise Environment**
- Hands-On Exercise: Installing the Kafka Service with Cloudera Manager
- Optional Hands-On Exercise: Create Metrics Dashboards
- Optional Hands-On Exercise: Using the CM API
- Essential Points

# Hands-On Exercise: Preparing the Exercise Environment

---

- In this exercise, you will access and prepare the exercise environment for installing the Kafka service.
  - Access the VM using the link provided by your instructor.
  - Read the General Notes.
  - Perform the steps under Prerequisites to set environment variables.
  - Optionally import exercise code into Eclipse
  - Optionally configure terminal titles.

# Chapter Topics

---

## Deploying Apache Kafka

- System Requirements and Dependencies
- Service Roles
- Planning Your Deployment
- Deploying Kafka Services
- Hands-On Exercise: Preparing the Exercise Environment
- **Hands-On Exercise: Installing the Kafka Service with Cloudera Manager**
- Optional Hands-On Exercise: Create Metrics Dashboards
- Optional Hands-On Exercise: Using the CM API
- Essential Points

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

---

- In this exercise, you will install Kafka, Streams Messaging Manager, Schema Registry, and ZooKeeper services on a 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: Preparing the Exercise Environment
- Hands-On Exercise: Installing the Kafka Service with Cloudera Manager
- **Optional Hands-On Exercise: Create Metrics Dashboards**
- Optional Hands-On Exercise: Using the CM API
- Essential Points

## Optional Hands-On Exercise: Create Metrics Dashboards

---

- In this exercise, will import dashboards into Cloudera Manager and view the associated charts.
- Exercise directory: `exercise-code/create-dashboards`

# Chapter Topics

---

## Deploying Apache Kafka

- System Requirements and Dependencies
- Service Roles
- Planning Your Deployment
- Deploying Kafka Services
- Hands-On Exercise: Preparing the Exercise Environment
- Hands-On Exercise: Installing the Kafka Service with Cloudera Manager
- Optional Hands-On Exercise: Create Metrics Dashboards
- **Optional Hands-On Exercise: Using the CM API**
- Essential Points

## Optional Hands-On Exercise: Using the CM API

---

- In this exercise, you will install and use the Cloudera Manager Python REST API.
- Exercise directory: `exercise-code/cm-api`

# Chapter Topics

---

## Deploying Apache Kafka

- System Requirements and Dependencies
- Service Roles
- Planning Your Deployment
- Deploying Kafka Services
- Hands-On Exercise: Preparing the Exercise Environment
- Hands-On Exercise: Installing the Kafka Service with Cloudera Manager
- Optional Hands-On Exercise: Create Metrics Dashboards
- Optional Hands-On Exercise: Using the CM API
- **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 CDP services

# Bibliography

---

The following offer more information on topics discussed in this chapter

- [CDP Private Cloud Base Requirements and Supported Versions](#)
- [CDP Private Cloud Base: Kafka Installation](#)
- [Cloudera Reference Architecture Documentation](#)
  - [CDP Data Center "Private Cloud Base" Reference Architecture](#)



## Kafka Command Line Basics

---

Chapter 4

# Course Chapters

---

- Introduction
- Kafka Overview
- Deploying Apache Kafka
- **Kafka Command Line Basics**
- Using Streams Messaging Manager (SMM)
- Kafka Java API Basics
- Improving Availability through Replication
- Improving Application Scalability
- Improving Application Reliability
- Analyzing Kafka Clusters with SMM
- Monitoring Kafka
- Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka Cluster
- Conclusion

# Kafka Command Line Basics

---

**By the end of this chapter, you will be able to**

- **Use the command-line interface to create and manage Kafka topics**
- **Publish messages and read messages from the command line**

# Chapter Topics

---

## Kafka Command Line Basics

- **Create and Manage Topics**
- Running Producers and Consumers
- Essential Points

# Creating Topics from the Command Line

---

- Kafka includes a convenient set of command line tools
  - These are helpful for exploring and experimentation
- The `kafka-topics` command offers a simple way to create Kafka topics
  - Provide the topic name of your choice, such as `device_status`
  - You must also specify one or more Kafka brokers
  - Older versions of Kafka used a list of ZooKeeper instances instead of Kafka brokers

```
$ kafka-topics --create \  
--bootstrap-server brokerhost1:9092,brokerhost2:9092 \  
--replication-factor 3 \  
--partitions 5 \  
--topic device_status
```

# Displaying Topics from the Command Line

---

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

```
$ kafka-topics --list \
--bootstrap-server cmhost:9092
```

- Use the `--help` option to list all `kafka-topics` options

```
$ kafka-topics --help
```

# Writing File Contents to Topics Using the Command Line

---

- **Using UNIX pipes or redirection, you can read input from files**
  - The data can then be sent to a topic using the command line producer
- **This example shows how to read input from a file named alerts.txt**
  - Each line in this file becomes a separate message in the topic

```
$ cat alerts.txt | kafka-console-producer \
--broker-list brokerhost1:9092,brokerhost2:9092 \
--topic device_status
```

- **This technique can be an easy way to integrate with existing programs**

# 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**

## 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

- Create and Manage Topics
- **Running Producers and Consumers**
- Essential Points

# CLI Tools for Producing and Consuming Messages

---

- Kafka provides command-line utilities for producing and consuming messages
  - `kafka-console-producer` can be used to send data from a console to a topic
  - `kafka-console-consumer` can be used to view messages from a particular topic
- Commands are typically used for inspection / testing (not for production use)

# Running a Producer from the Command Line (1)

---

- You can run a producer using the `kafka-console-producer` tool
- Specify one or more brokers in the `--broker-list` option
  - Each broker consists of a hostname, a colon, and a port number
  - If specifying multiple brokers, separate them with commas
- You must also provide the name of the topic

```
$ kafka-console-producer \
--broker-list brokerhost1:9092,brokerhost2:9092 \
--topic device_status
```

## Running a Producer from the Command Line (2)

---

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

# Running a Consumer from the Command Line

---

- You can run a consumer with the `kafka-console-consumer` tool
- Requires one or more bootstrap servers (Kafka brokers)
  - Older versions used ZooKeeper instances instead
- The command also requires a topic name
- Use `--from-beginning` to read *all* available messages
  - Otherwise, it reads only new messages

```
$ kafka-console-consumer \
--bootstrap-server brokerhost:9092 \
--topic device_status \
--from-beginning
```

# Chapter Topics

---

## Kafka Command Line Basics

- Create and Manage Topics
- Running Producers and Consumers
- Essential Points

## Essential Points

---

- **Kafka command-line utilities are**
  - Used to easily create and manage topics, producers, and consumers
  - Require access to Kafka or Zookeeper
  - Typically used for inspection / testing
- **Setting environment variables for hostnames is recommended**

# Bibliography

---

The following offer more information on topics discussed in this chapter

- [Kafka Topic Naming Conventions](#)
- [Apache Kafka Command Line Tools](#)



# Using Streams Messaging Manager (SMM)

---

Chapter 5

# Course Chapters

---

- Introduction
- Kafka Overview
- Deploying Apache Kafka
- Kafka Command Line Basics
- **Using Streams Messaging Manager (SMM)**
- Kafka Java API Basics
- Improving Availability through Replication
- Improving Application Scalability
- Improving Application Reliability
- Analyzing Kafka Clusters with SMM
- Monitoring Kafka
- Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka Cluster
- Conclusion

# Using SMM to Monitor Kafka Clusters

---

By the end of this chapter, you will be able to

- Identify and monitor producers and consumers for a topic or broker
- View end-to-end flow of message streams
- Create and manage topics

# Chapter Topics

---

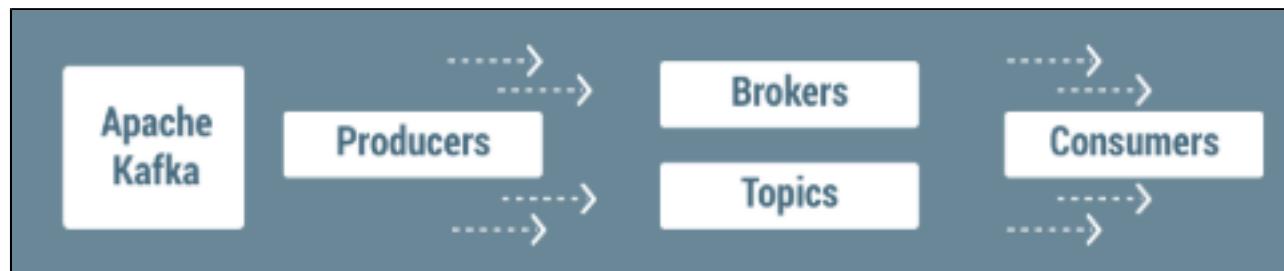
## Using Streams Messaging Manager (SMM)

- **Streams Messaging Manager Overview**
- Producers, Topics, and Consumers
- Data Explorer
- Brokers
- Topic Management
- Hands-On Exercise: Managing Topics using the CLI
- Hands-On Exercise: Connecting Producers and Consumers from the Command Line
- Essential Points

# Cloudera Streams Messaging Manager

---

- Cloudera Streams Messaging Manager is an operations monitoring and management tool for Kafka clusters
  - Gain insights about your Kafka clusters
  - Understand the end-to-end flow of message streams
  - Identify bottlenecks, throughputs, and traffic flow
  - Optimize your Kafka environment based on key performance insights



# Streams Messaging Manager Benefits

- Provides a dashboard to visually monitor all of your Kafka clusters
- Cures "Kafka Blindness"
  - Struggle to understand what is happening inside Kafka
  - Difficulty knowing the state of topics and brokers
  - Inability to see who is producing and consuming the data
  - Difficulty monitoring and troubleshooting the flow of data

The screenshot shows the Streams Messaging Manager (SMM) Overview dashboard for a cluster named 'CDFCluster'. The top navigation bar includes tabs for 'Overview', 'Producers', 'Brokers', 'Topics', 'Consumer Groups', and 'Alerts'. The main area displays four key metrics: Producers (35 of 101), Brokers (3 of 3), Topics (4 of 50), and Consumer Groups (3 of 21). Below these are two tables: 'Producers' and 'Consumer Groups'. The 'Producers' table lists 18 producers with names like 'minifi-eu-i4' through 'minifi-eu-i18', their replication factor (2), in-sync replicas (6 of 6), total messages (1,338), and retention period (0 secs). The 'Consumer Groups' table lists three groups: 'nifi-truck-sensors-east' (lag 5), 'nifi-truck-sensors-west' (lag 2), and 'nifi-truck-sensors-central' (lag 2). A sidebar on the left provides links to 'Topics', 'Producers', 'Consumer Groups', and 'Alerts'.

## Designed for the Enterprise

---

- Single monitoring dashboard
- Support for multiple Kafka clusters
- Intelligent filtering
- Topic management
- Alert policies and notifications
- Authentication and authorization

# SMM Components

---

- **SMM consists of two components**
  - SMM UI Server
  - SMM REST Admin Server
- **SMM UI powered by first class REST Services via the SMM REST Admin Server**

# SMM REST API

- REST as a "first class citizen"
- Extends the monitoring and management capabilities
- Can be used to integrate with APM/alerting/ticketing solutions

The screenshot shows the SMM REST API documentation interface. It is divided into two main sections:

- Alert notifications**: This section contains several API endpoints:
  - GET /api/v1/admin/alert/notifications/entity/{entityType}/{entityName}**: Get the Alert notifications for the given entity type and resource name.
  - GET /api/v1/admin/alert/notifications**: Get the Alert notifications from the given 'offset' up to the given 'limit'. If the offset is '1' it will return 'limit' number of latest notifications. If the offset is '2' it will return 'limit' number of notifications starting from 2nd offset.
  - GET /api/v1/admin/alert/notifications/entity/{entityType}**: Get the Alert notifications for the given entity type.
  - PUT /api/v1/admin/alert/notifications/read**: Mark the Alert notification as read.
  - PUT /api/v1/admin/alert/notifications/unread**: Mark the alert notification as unread.
- Consumer group related details.**: This section contains several API endpoints:
  - GET /api/v1/admin/consumers/groupNames**: Get all consumer group names.
  - GET /api/v1/admin/consumers/groups**: Get all consumer group details.
  - GET /api/v1/admin/consumers/groups/{groupName}**: Get consumer group details for the given groupName 'groupName'.
  - GET /api/v1/admin/consumers/clients**: Get consumer group details for all clients.
  - GET /api/v1/admin/consumers/clients/{clientId}**: Get consumer group details for a given clientId.

# Chapter Topics

---

## Using Streams Messaging Manager (SMM)

- Streams Messaging Manager Overview
- **Producers, Topics, and Consumers**
- Data Explorer
- Brokers
- Topic Management
- Hands-On Exercise: Managing Topics using the CLI
- Hands-On Exercise: Connecting Producers and Consumers from the Command Line
- Essential Points

# Monitor Topics

---

## Questions

What producers and consumers are connected to a topic?

What are the topic's retention rate?

What are the total messages going into a topic?

Are my replicas in sync?

How are the partitions laid out for the topic?

- **SMM displays all topics in your cluster**
  - Filters producers and consumers relevant for the selected topic
  - Displays data in and data out for each partition
  - Displays values for incoming messages, retention rate, and replica information

# Display Topics

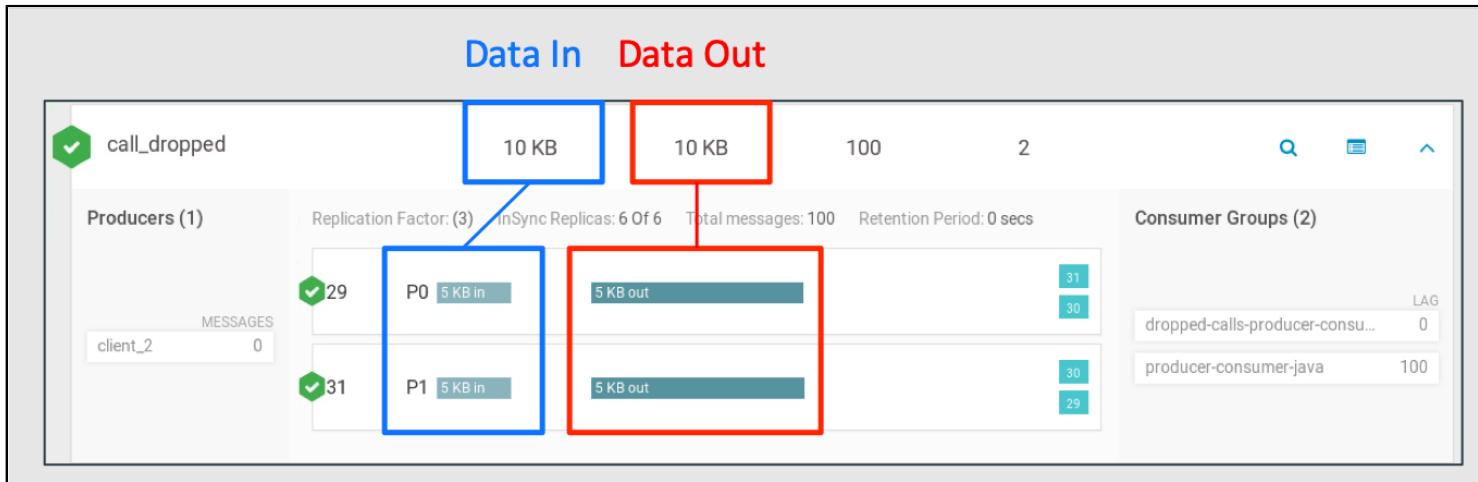
- The **Topics** page lists all topics in your cluster
- Metric totals for all topics displayed on top
- Use the search box to filter on a subset
- Specify time duration using the dropdown

The screenshot shows the Cloudera Manager Topics page for Cluster 1. At the top, there are two search/filter sections: "Search topics" with a search bar containing "cal" and a dropdown for "Select duration" set to "24 hours". Below these are metric summary values: Total Bytes In (228 B), Total Bytes Out (4 MB), Produced Per Sec (0), Fetched Per Sec (760), In Sync Replicas (140), Out Of Sync (69), Under Replicated (66), and Offline Partitions (0). A blue arrow points from the "Search topics" section to the search bar. Another blue arrow points from the "Select duration" section to the dropdown menu.

NAME	DATA IN	DATA OUT	MESSAGES IN	CONSUMER GROUPS	Actions
call_dropped	0B	0B	0	0	
call_test	0B	0B	0	0	
call_failed	0B	0B	0	0	
call_success	0B	3 KB	0	0	

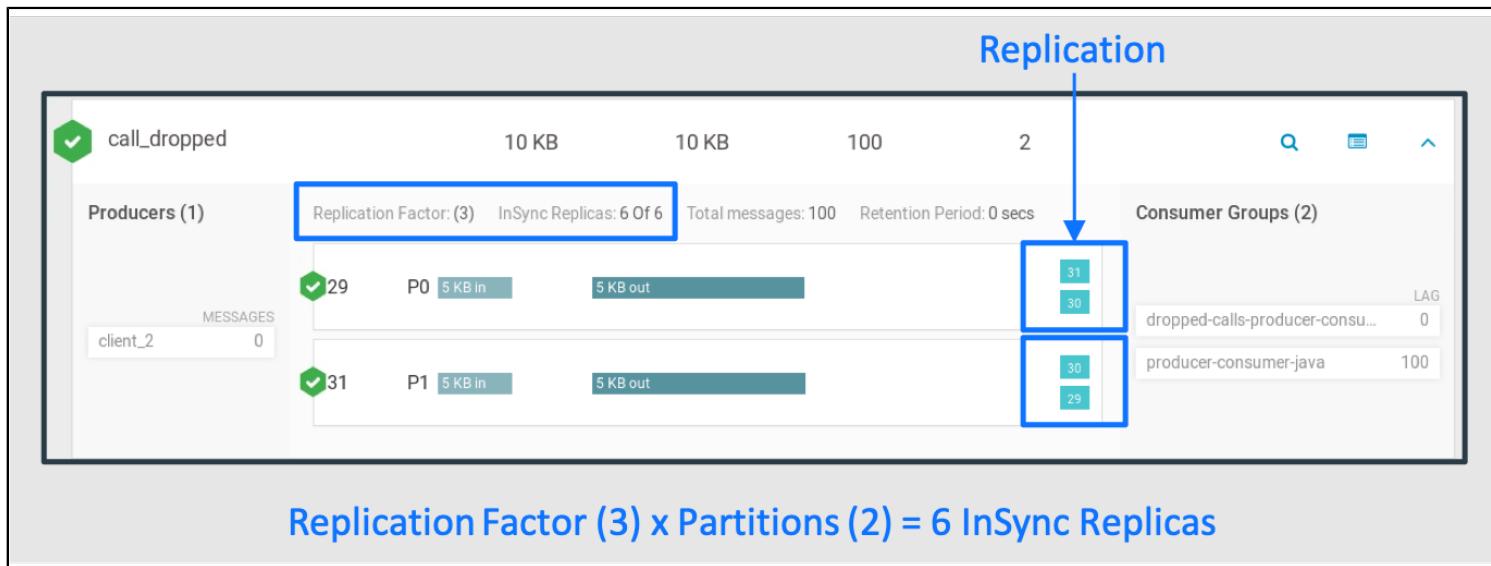
# Topic Partitions

- Each topic can be configured with one or more partitions
- Selected topic displays amount of data in and data out for each partition



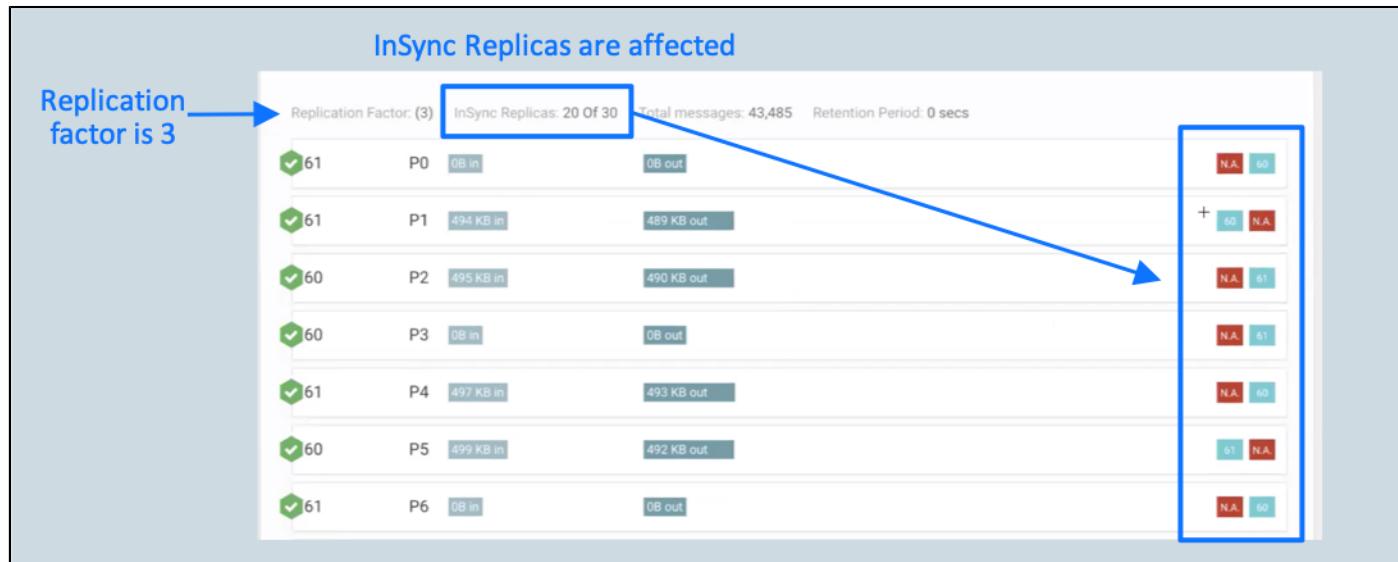
# Topic Metrics

- Replication Factor
- In-sync Replicas
- Total Messages
- Retention Period



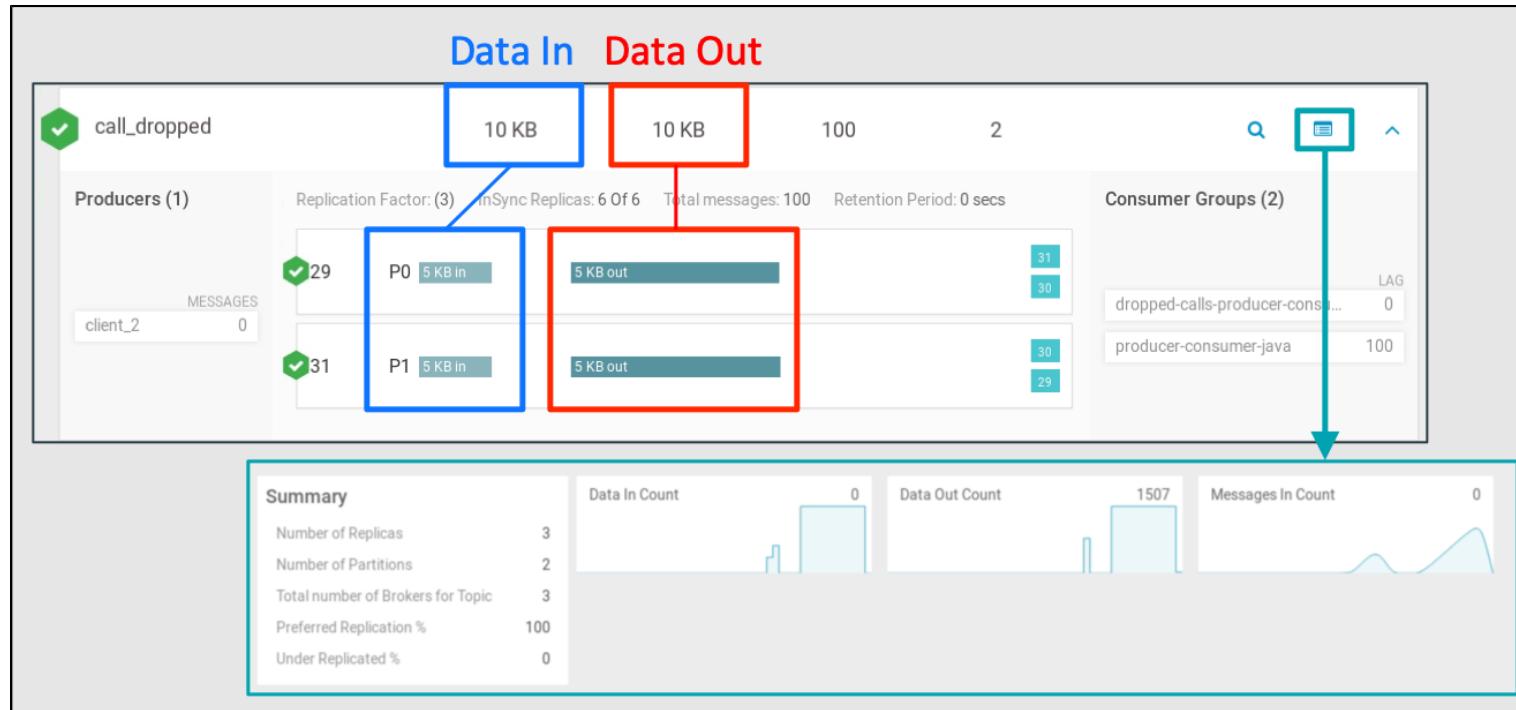
# Replication Issues

- Red highlighted N/A indicates that replicas are not responding



# Topic Profile

- Data in and data out totals are displayed for each topic partition
- Click the **Profile** icon to view metric details
  - Summary
  - Data In Count and Data Out Count
  - Messages In count



# Topics Overview

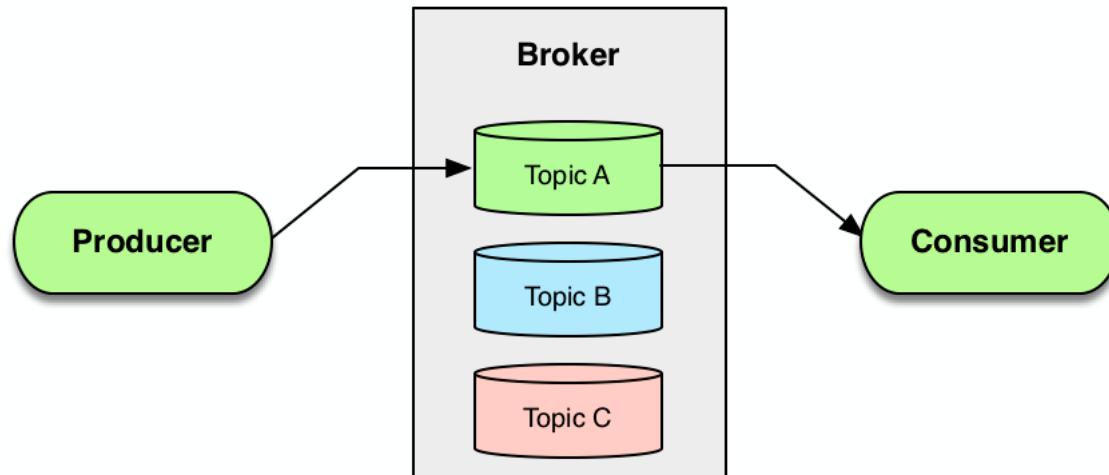
The screenshot shows the Apache Kafka Streams Metrics UI Overview page. At the top, there are dropdown menus for Producers (35 of 101), Brokers (3 of 3), Topics (4 of 50), and Consumer Groups (3 of 21). A blue box highlights the 'Topics' dropdown. Below it, a table lists topics with columns for Name, Size, and other metrics. A modal window is open for the topic 'gateway-west-raw-sensors', showing replication factor, log offsets, and message counts. To the right, a sidebar displays consumer groups with their respective lag counts. A search bar at the top right is set to 'call'. A blue arrow points from the 'Topics' dropdown to the search bar. Another blue arrow points from the search bar to the 'Time duration' dropdown, which is set to '30 minutes'.

- The **Overview page** displays Kafka entities
  - Filter on topics using the **Topics** dropdown menu
  - Search for topics using the search box
  - Each topic can be expanded to view metric details
  - Lists same information as on the **Topics** page
- The **Overview page** also displays producers and consumers, which will be discussed next

# Producers and Consumers

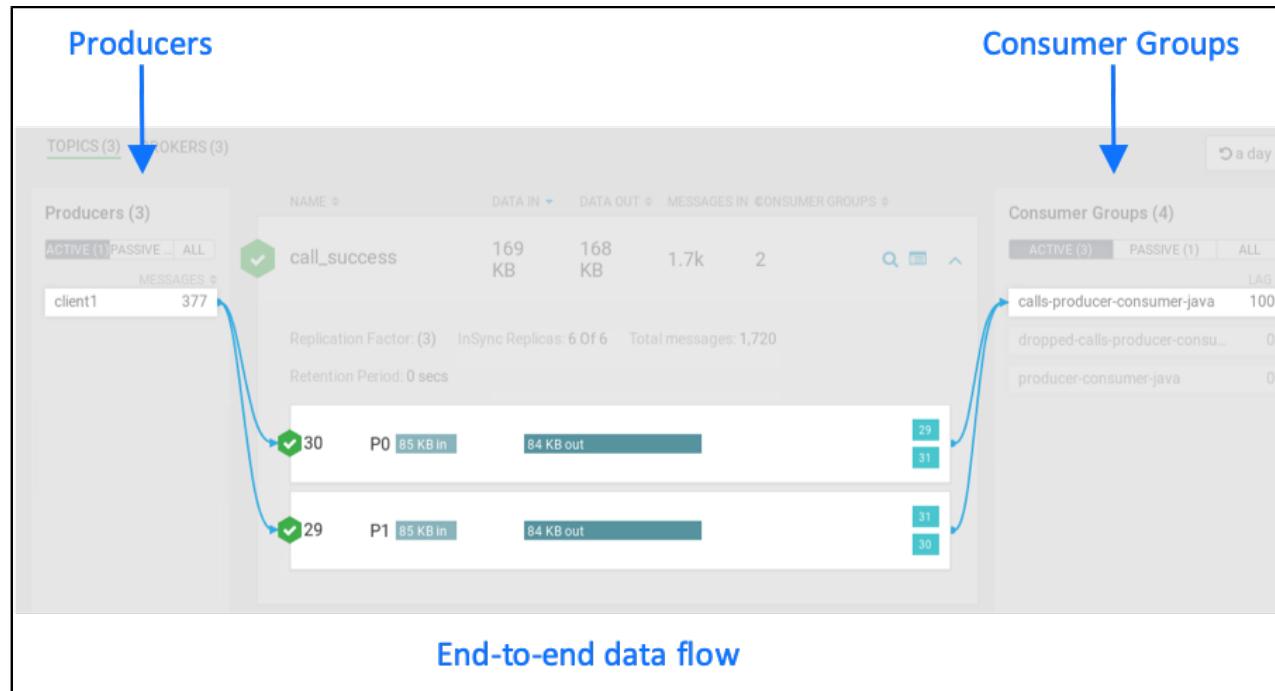
## Questions

- Which producers are feeding which topics?
- Which consumers are consuming from which topics?
- What are my consumer groups?

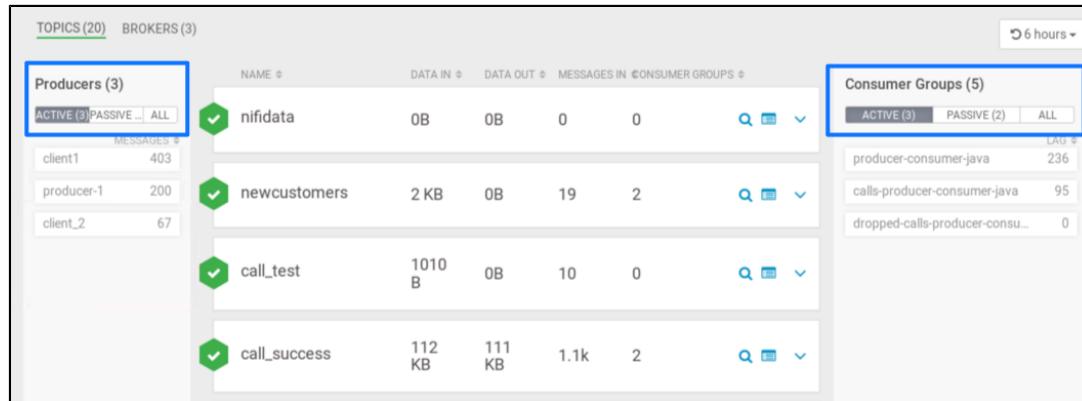


# Producers and Consumers Data Flow

- Overview page displays end-to-end data flow from producers to consumer groups
  - What producers publish to which topics
  - What consumer groups consume from which topics
- Can also filter and search for specific producers and consumer groups



# Active Versus Passive



- **Producers and consumers display as active or passive**
  - Producers are active when they are producing messages over a designated time period
  - Similarly, consumers are active when they are consuming messages over a designated time period
  - Otherwise, producers and consumers display as passive
- **Time period is specified in the topic's advanced configuration**
  - `inactive.producer.timeout.ms` (default is 1800000 ms)
  - `inactive.group.timeout.ms` (default is 1800000 ms)

# Viewing Producers

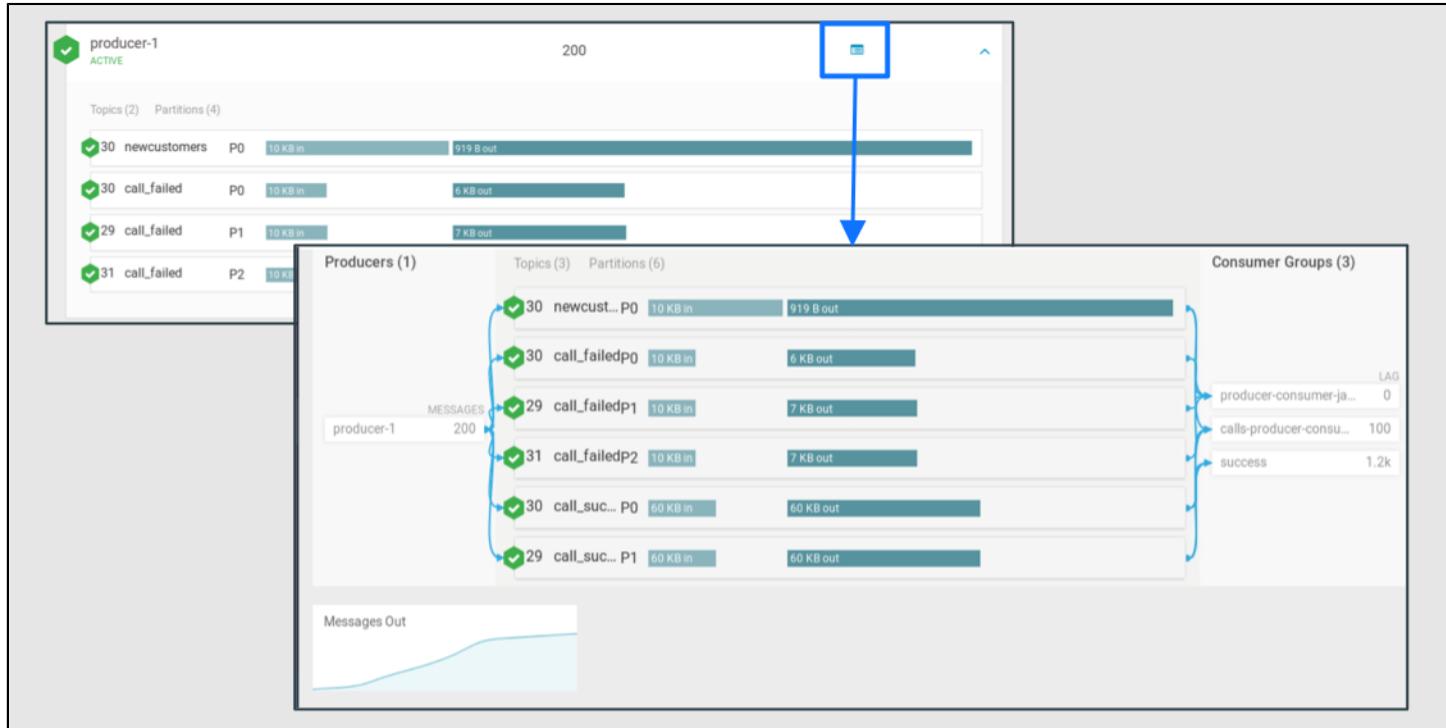
- Producer name in SMM is based on the producer's `client.id` property
- Click on **Messages** to sort on messages sent within the specified duration
- Producers page provides metrics

The screenshot illustrates the Cloudera Manager interface for viewing producers. On the left, a sidebar shows a summary of 'Producers (3)'. The main area, titled 'Producers Page', displays three active producers: 'producer-1', 'client\_2', and 'client1'. Each producer entry includes a status icon (green checkmark for active), the producer name, and the number of messages sent (200, 67, and 490 respectively). A blue arrow points from the 'producer-1' row to a detailed view of its metrics. This detailed view shows 'Topics (2)' and 'Partitions (4)'. For topic 'newcustomers', partitions P0 and P1 have 10 KB in and 919 KB out respectively. For topic 'call\_failed', partitions P0, P1, and P2 have 10 KB in and 6 KB, 7 KB, and 7 KB out respectively.

Topic	Partition	In (KB)	Out (KB)
newcustomers	P0	10 KB	919 KB
	P1	10 KB	7 KB
call_failed	P0	10 KB	6 KB
	P1	10 KB	7 KB
	P2	10 KB	7 KB

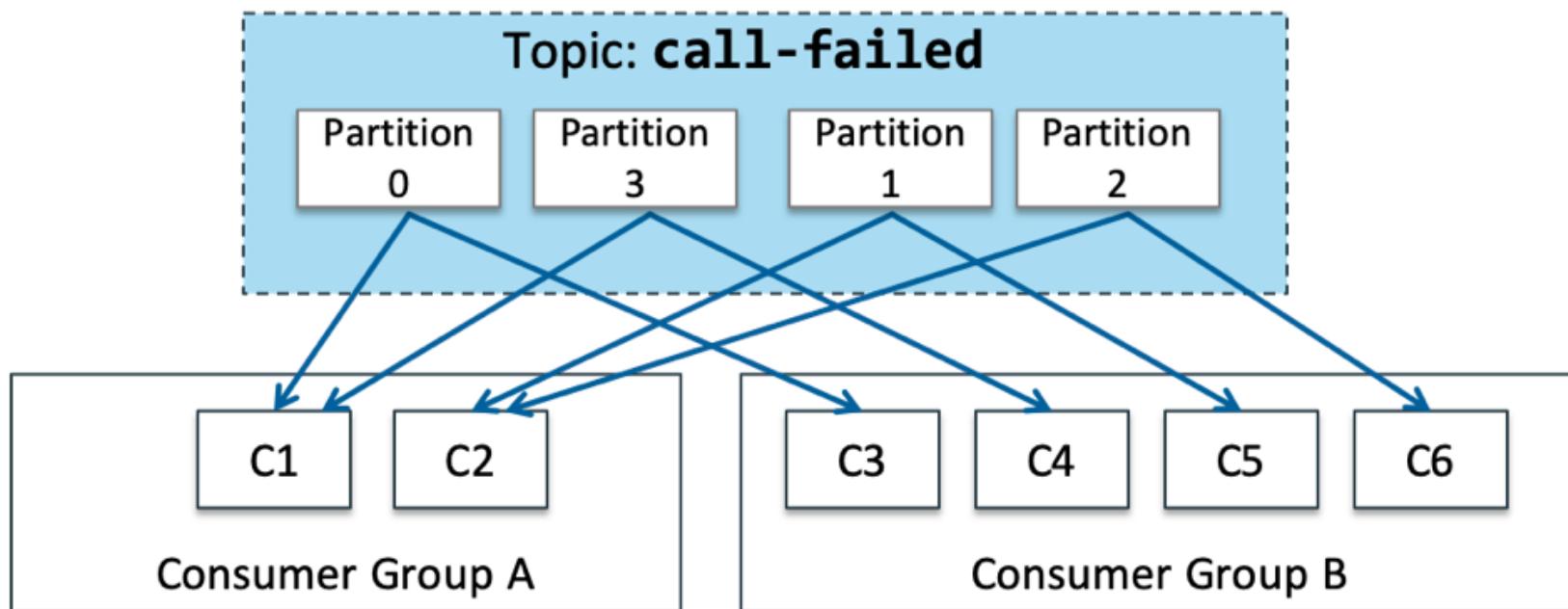
# Producer Profile

- Click the Profile icon to display data flow
- Similar display is provided on the Overview page



# 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
  - Each message published to a topic is delivered to one consumer instance within each subscribing consumer group
  - Message ordering is preserved per partition, but not across the topic



# Viewing Consumer Groups

## ■ Consumer Groups Page

- Sort ascending or descending by **NAME** or **LAG**
- Status is **ACTIVE** or **INACTIVE(PASSIVE)**
- Lag indicates the number of messages that the consumer is behind on consuming

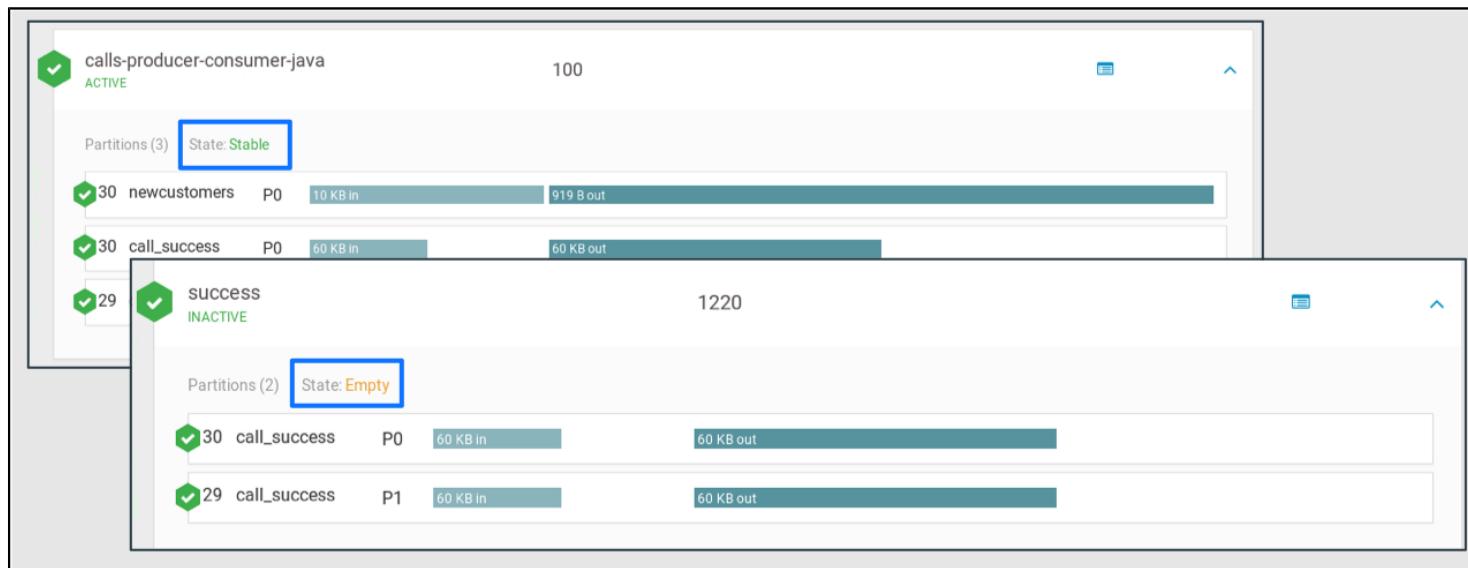
The screenshot shows the 'Consumer Groups' page in the Cloudera Manager interface. The page title is 'Consumer Groups Page' and it specifies 'Cluster: Cluster 1'. On the left, there is a sidebar with a 'Consumer Groups (5)' section containing three tabs: 'ACTIVE (3)', 'PASSIVE (2)', and 'ALL'. Below the tabs, there is a table with two rows: 'calls-producer-consumer-java' (LAG: 100) and 'dropped-calls-producer-consu...' (LAG: 0). A blue arrow points from this sidebar area to the main table on the right.

NAME	LAG	Action
__smm-app INACTIVE	0	[Edit] [Delete]
dropped-calls-producer-consumer-java ACTIVE	0	[Edit] [Delete]
success INACTIVE	1220	[Edit] [Delete]
producer-consumer-java ACTIVE	0	[Edit] [Delete]
calls-producer-consumer-java ACTIVE	100	[Edit] [Delete]

# Consumer Group States

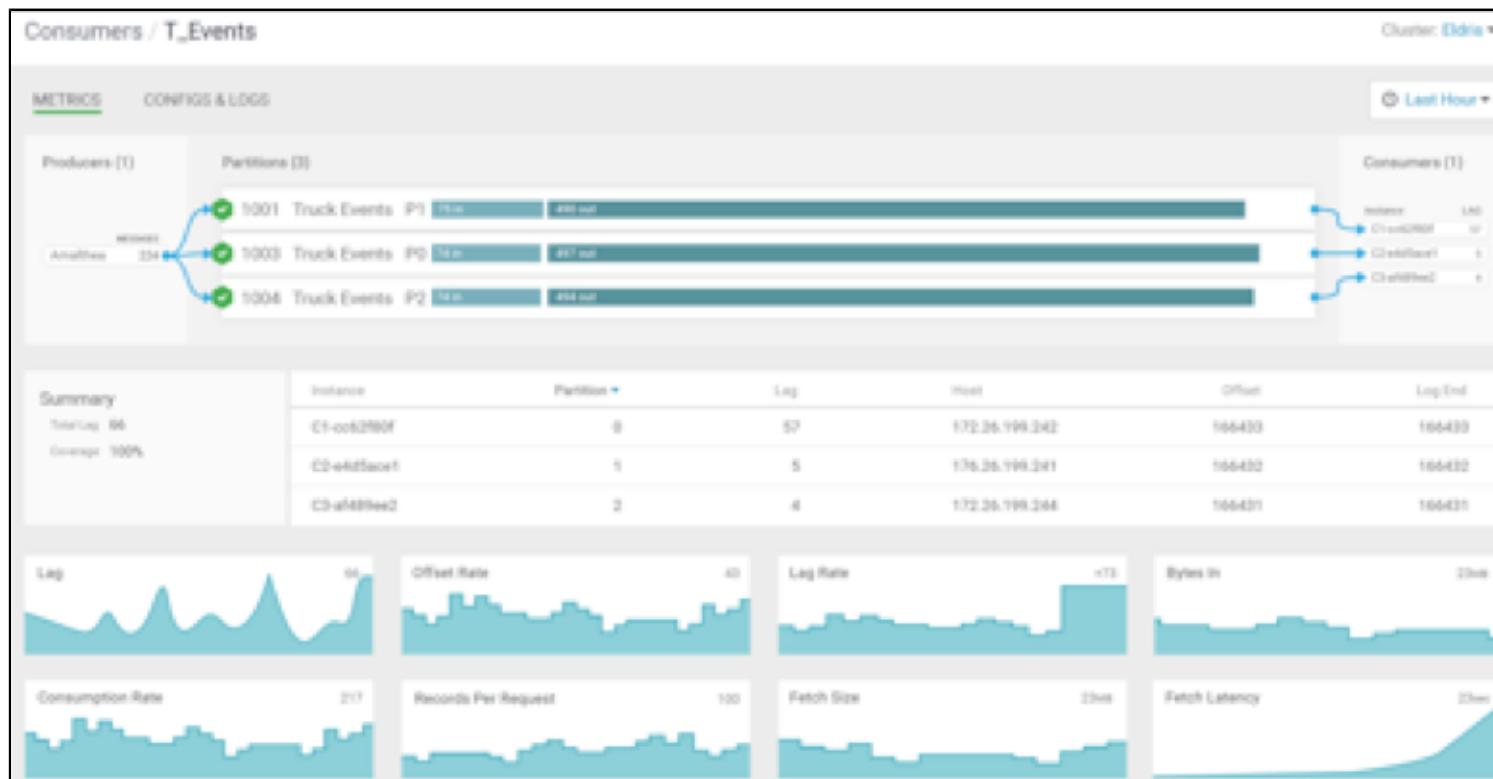
## ■ Consumer group state

- **Empty:** The group exists but has no consumers
- **Stable:** Consumers are consuming from associated topics
- **PreparingRebalance:** Kafka is preparing to rebalance the group
- **CompletingRebalance:** Kafka is still rebalancing the group
- **Dead:** The group is going to be removed soon (for example, due to inactivity)



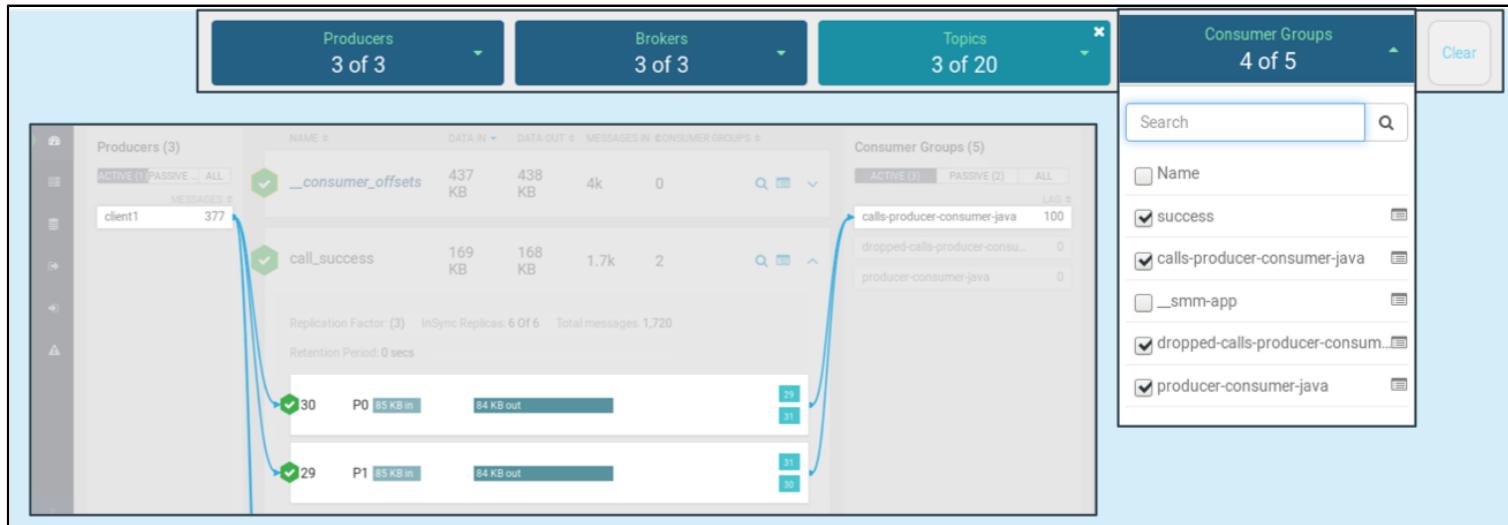
# Consumer Group Profile

- The consumer group profile displays
  - Number of consumers in the group
  - Details about lag and offsets



# Data Flow Summary

- The Overview page displays end-to-end message streams
- Filter and search on producers, brokers, topics, and consumers to isolate relevant data flows



# Chapter Topics

---

## Using Streams Messaging Manager (SMM)

- Streams Messaging Manager Overview
- Producers, Topics, and Consumers
- **Data Explorer**
- Brokers
- Topic Management
- Hands-On Exercise: Managing Topics using the CLI
- Hands-On Exercise: Connecting Producers and Consumers from the Command Line
- Essential Points

# Data Explorer

---

## Questions

What is happening with topic messages on each partition?

When did the message get into the topic?

What is the content of each message?

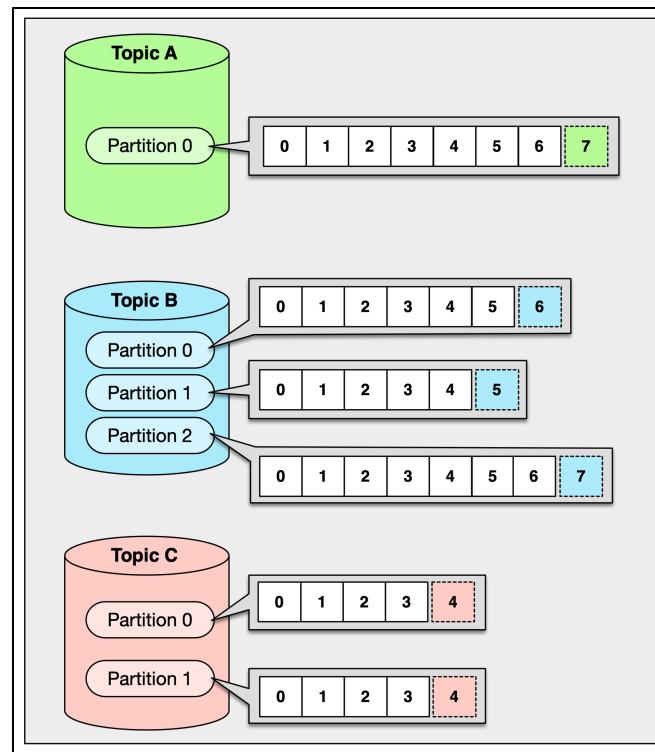
What is the offset?

- Click the Data Explorer icon to view topic data
  - Displays messages and associated offset on each partition
  - View message content for formats that are readable

## Review: Partitions

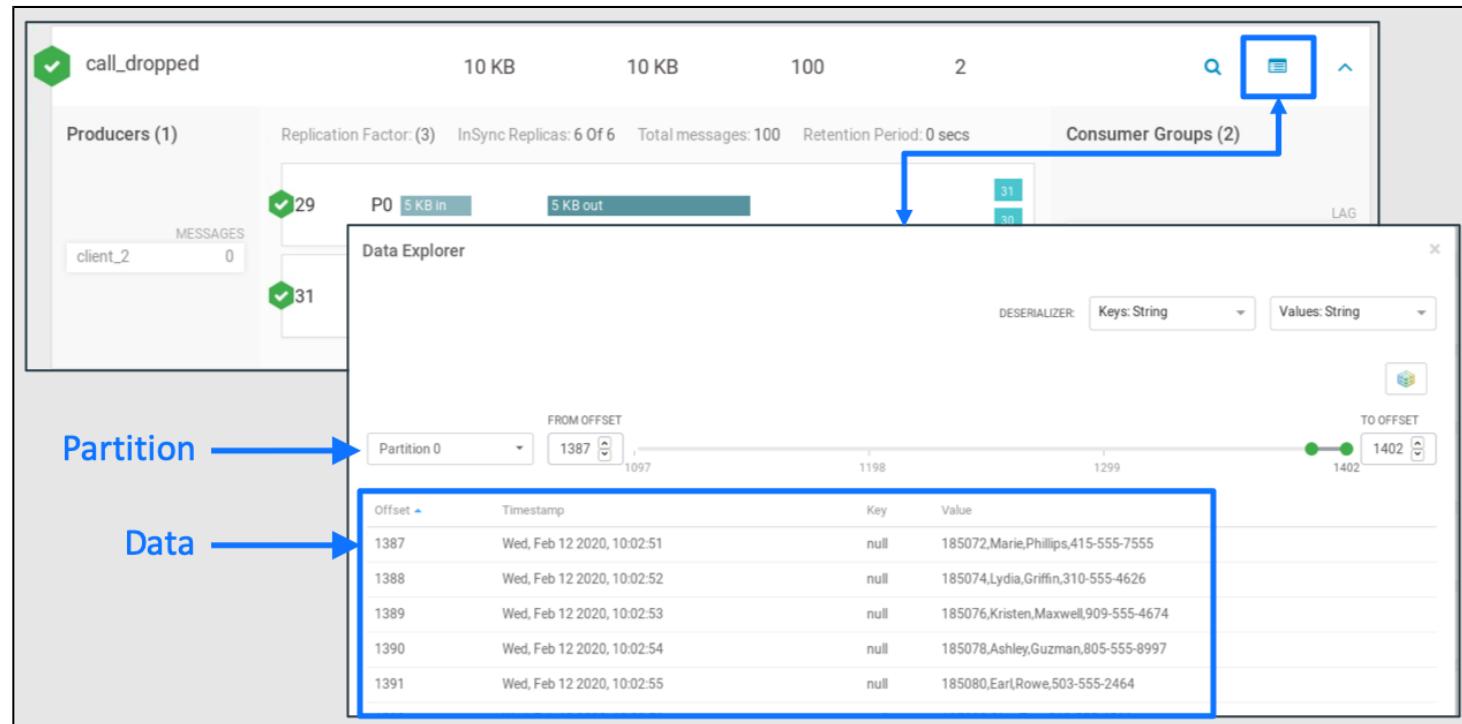
---

- Partitions allow you to parallelize a topic by splitting the data in a particular topic across multiple brokers
- Each partition can be placed on a separate machine to allow for multiple consumers to read from a topic in parallel



# Data by Partition

- Click the Data Explorer icon from the topic
- Select partition using the Partition dropdown menu



# Data Types

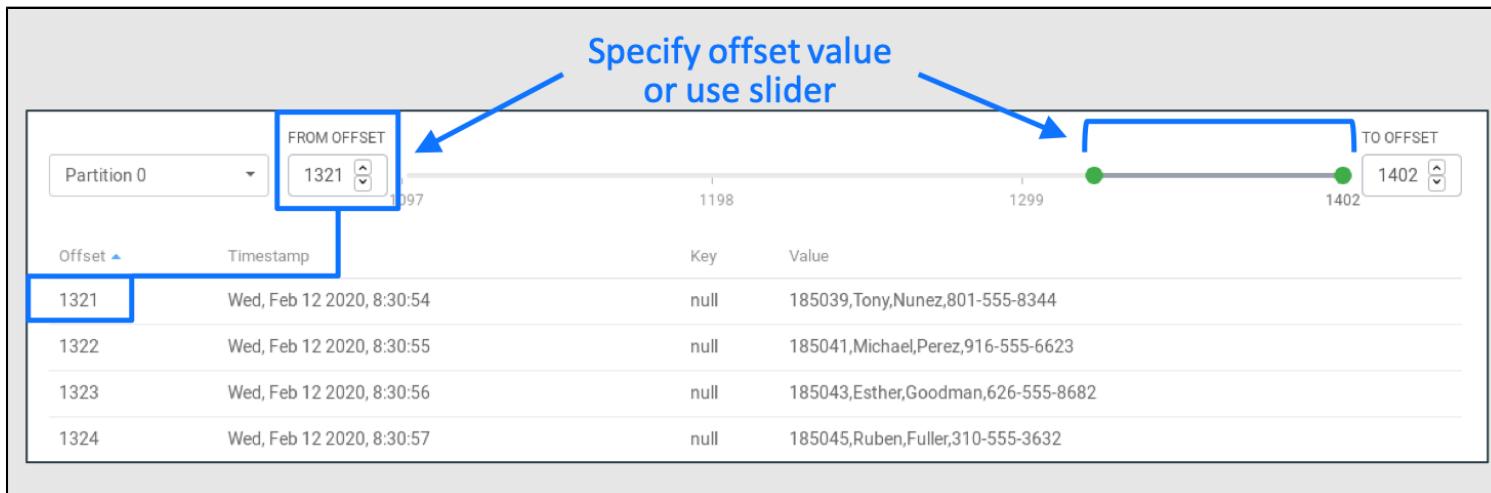
The screenshot shows the Data Explorer interface. At the top, there are dropdown menus for 'DESERIALIZER' set to 'Keys: String' and 'Values: String'. Below this, a 'FROM OFFSET' section allows selecting a partition (Partition 0) and an offset (1325). A scrollable table displays two messages with columns: Offset, Timestamp, Key, and Value. The first message at offset 1325 has a timestamp of 'Wed, Feb 12 2020, 8:30:58' and a value of '185047,Victor,Berry,619-555-3982'. The second message at offset 1326 has a timestamp of 'Wed, Feb 12 2020, 8:30:59' and a value of '185049,Tony,Hansen,310-555-9743'. To the right of the table, a vertical list of deserializer types is shown: String, Short, Integer, and Long.

Offset	Timestamp	Key	Value
1325	Wed, Feb 12 2020, 8:30:58	null	185047,Victor,Berry,619-555-3982
1326	Wed, Feb 12 2020, 8:30:59	null	185049,Tony,Hansen,310-555-9743

- Kafka message values include **String, Short, Integer, Long, Float, Double, Bytes, ByteArray, ByteBuffer, and Avro**
- Example message payload types
  - CSV
  - JSON
  - JPEGs
- Schema registry can be used for Avro

# Viewing Messages and Offset

- Specify the offset to filter on specific messages
  - Enter the value of the offset
  - Use the slider to specify a range
- Examine data on each partition using the Partition dropdown
  - Each partition will have messages with different offset values since messages are not duplicated; thus a consumer group will not get duplicate messages
  - If only one partition is specified, offset values will be set sequentially



# Chapter Topics

---

## Using Streams Messaging Manager (SMM)

- Streams Messaging Manager Overview
- Producers, Topics, and Consumers
- Data Explorer
- **Brokers**
- Topic Management
- Hands-On Exercise: Managing Topics using the CLI
- Hands-On Exercise: Connecting Producers and Consumers from the Command Line
- Essential Points

# Kafka Brokers

---

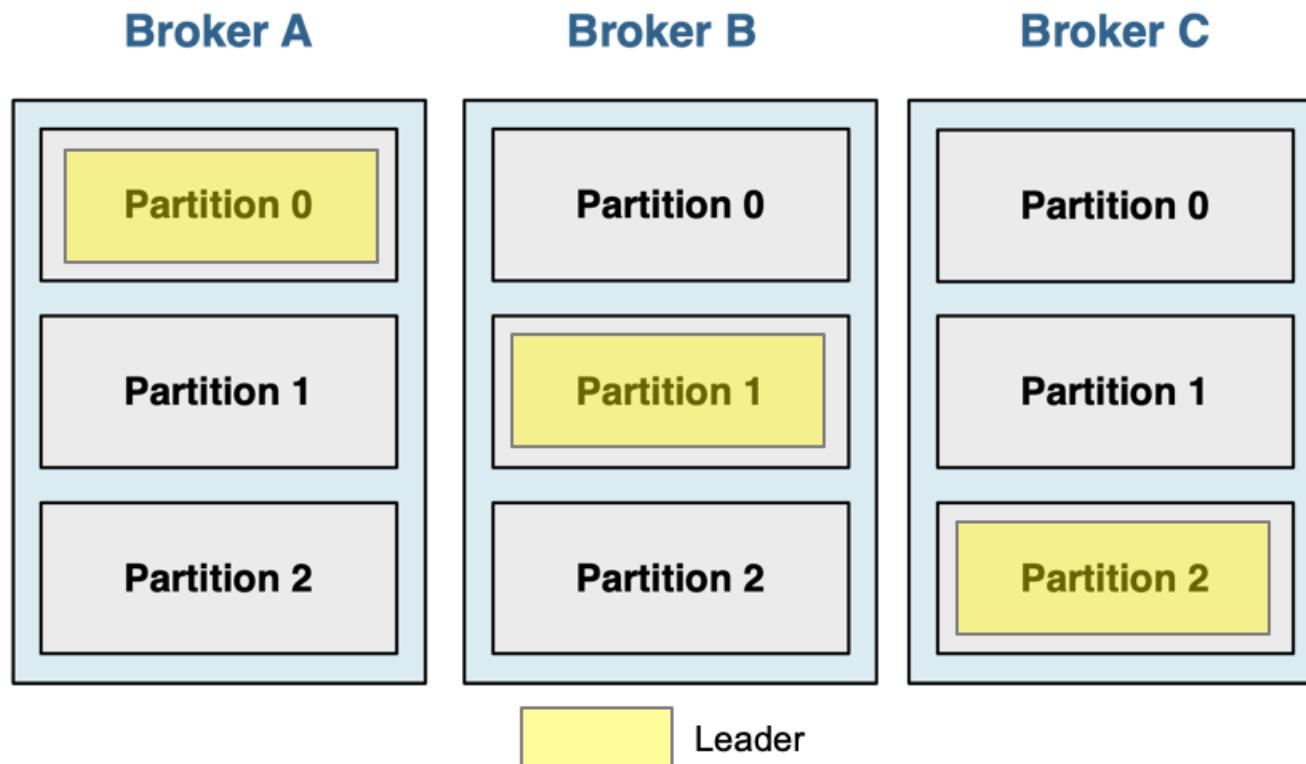
## Questions

- Are any of my brokers down?
- On what host is my broker located?
- Is my broker running out of disk space?
- Which partitions are located on each broker?
- What brokers hold the partitions for a specific topic?
- Are any of my brokers running hot?

# Kafka Review: Brokers

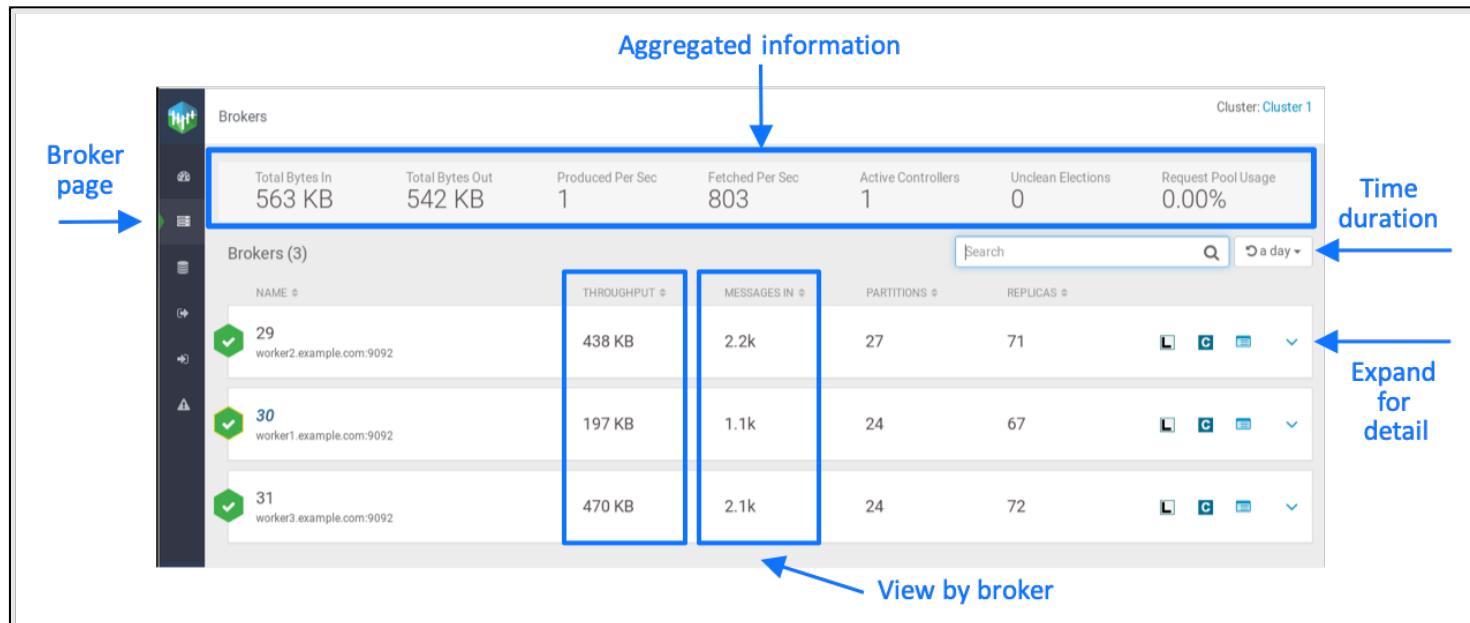
---

- Each host in the Kafka cluster runs a server called a broker that stores messages sent to the topics and serves consumer requests
- When creating a topic, you specify the replication factor and number of partitions



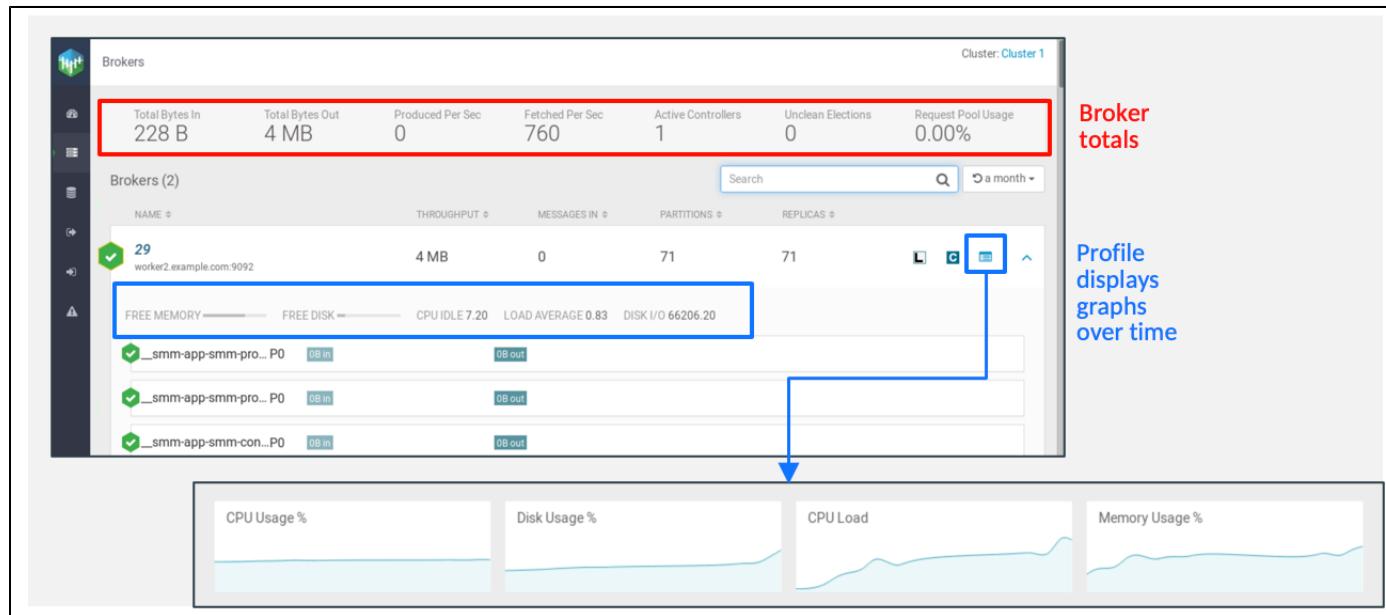
# View Brokers

- View brokers from the Brokers page
  - Total broker statistics displayed on top
  - Each broker displays throughput, messages in, partitions, and replica information
  - Expand arrow for broker detail



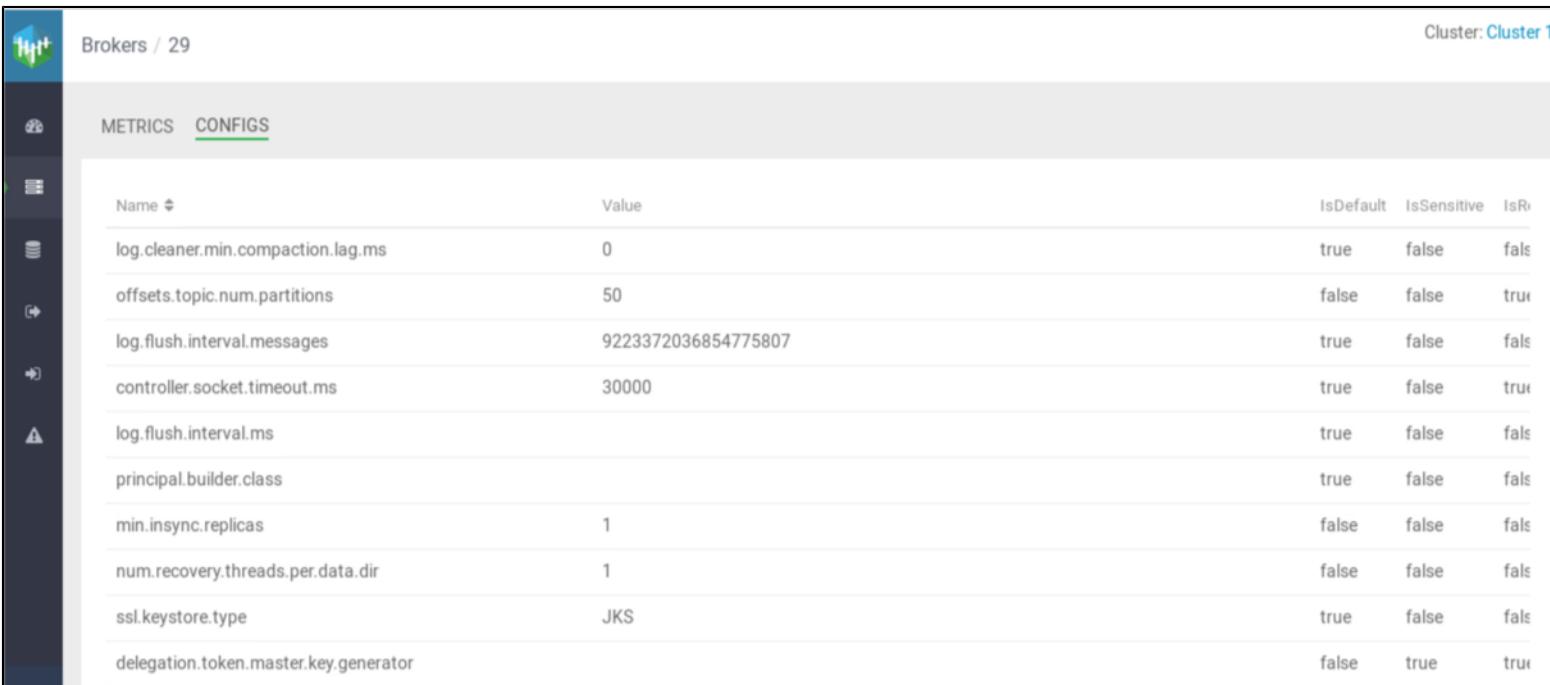
# Broker Metrics

- On the **Broker** page, expand a broker to display details
  - CPU Usage
  - Disk Usage
  - CPU Load
  - Memory Usage
- Click the **Profile** icon to view graphs over time



# Broker Configuration

- Select **CONFIGS** to view a broker's configuration settings
- Values are read only and cannot be updated in SMM



The screenshot shows the Cloudera Manager interface for managing brokers. The top navigation bar includes a logo, the text "Brokers / 29", and "Cluster: Cluster 1". On the left, there is a sidebar with various icons. The main content area has tabs for "METRICS" and "CONFIGS", with "CONFIGS" being active. Below the tabs is a table with the following data:

Name	Value	IsDefault	IsSensitive	IsR
log.cleaner.min.compaction.lag.ms	0	true	false	false
offsets.topic.num.partitions	50	false	false	true
log.flush.interval.messages	9223372036854775807	true	false	false
controller.socket.timeout.ms	30000	true	false	true
log.flush.interval.ms		true	false	false
principal.builder.class		true	false	false
min.insync.replicas	1	false	false	false
num.recovery.threads.per.data.dir	1	false	false	false
ssl.keystore.type	JKS	true	false	false
delegation.token.master.key.generator		false	true	true

# Chapter Topics

---

## Using Streams Messaging Manager (SMM)

- Streams Messaging Manager Overview
- Producers, Topics, and Consumers
- Data Explorer
- Brokers
- **Topic Management**
- Hands-On Exercise: Managing Topics using the CLI
- Hands-On Exercise: Connecting Producers and Consumers from the Command Line
- Essential Points

# Topic Management

---

- Create new topics
- Modify existing topics by updating configuration parameters
- Delete topics

# Create a Topic

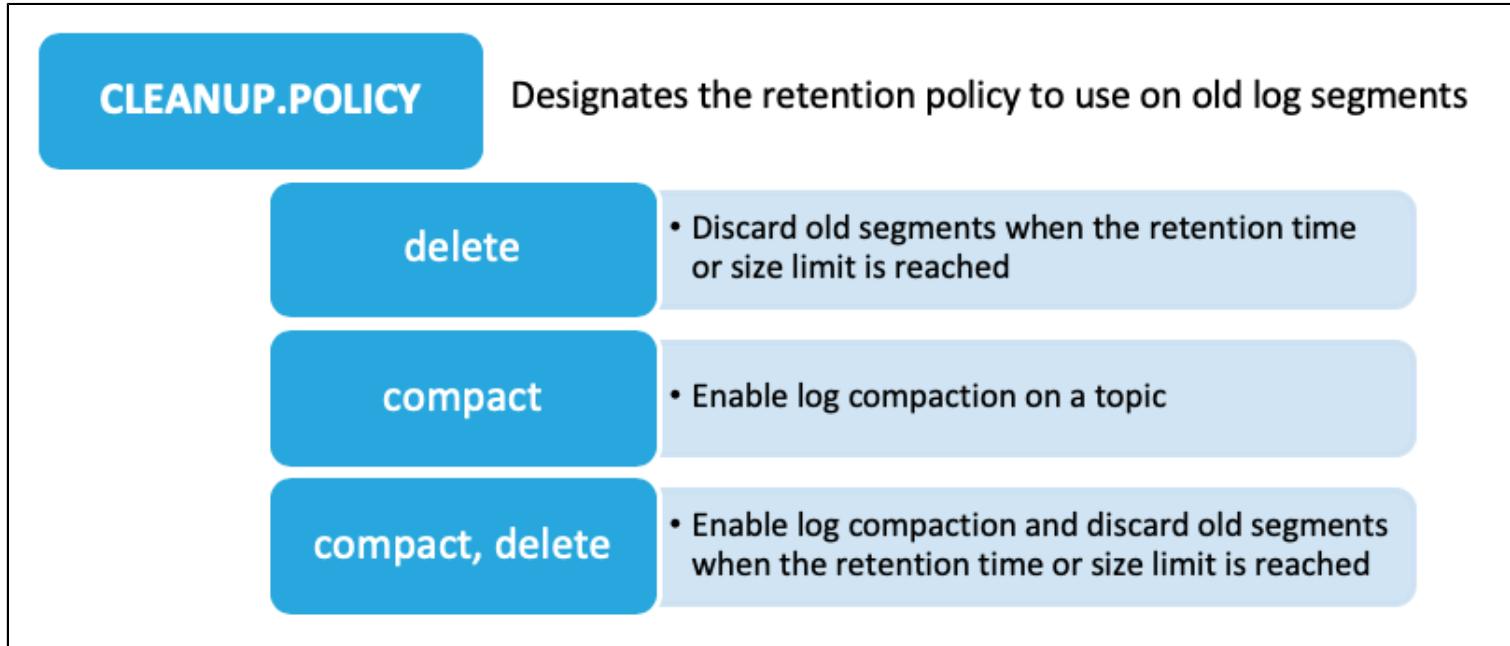
- Specify the topic name and number of partitions
- Select pre-defined availability or custom availability
- Select the cleanup policy

The screenshot shows the 'Add Topic' dialog in Cloudera Manager. In the 'Availability' section, five icons represent different replication levels: MAXIMUM (scales), HIGH (trophy), MODERATE (star), LOW (speaker), and CUSTOM (wrench). The CUSTOM icon is highlighted with an orange border and has an orange arrow pointing to a separate 'Custom Availability' dialog. The CUSTOM dialog contains fields for 'REPLICATION FACTOR' (set to 3) and 'MIN.INSYNC.REPLICAS' (set to 2).

Availability Level	Icon	Replication Factor	Min. InSync. Replicas
MAXIMUM	Scales	3	2
HIGH	Trophy	3	1
MODERATE	Star	2	1
LOW	Speaker	1	1
CUSTOM	Wrench	3	2

# Topic Cleanup Policy

- Messages on a topic are not immediately removed after they are consumed
- Cleanup policy specifies how the data expires



# Advanced Configuration

---

- Click Advanced to specify configuration parameters
- Displayed values are set to Kafka's default values
- **TOPIC NAME, PARTITIONS, and REPLICATION FACTOR** are required

Add Topic

TOPIC NAME	PARTITIONS
call_forward	3

REPLICATION FACTOR

3
---

CLEANUP.POLICY

delete
--------

MAX.MESSAGE.BYTES

1000000
---------

RETENTION.BYTES

-1
----

Simple Cancel Save

# Modify a Topic

- Click the Profile icon and select CONFIGS to view the configuration
- Click Advanced to modify topic configuration values
- Topic configuration values that cannot be modified in SMM
  - Topic name
  - Number of partitions
  - Replication factor

The screenshot shows the 'CONFIGS' tab of the topic configuration interface. At the top, it displays the topic name 'call\_dropped' and the number of partitions set to 2. Below this, there's a section for 'Availability' with five icons: 'MAXIMUM' (green), 'HIGH' (light green), 'MODERATE' (grey), 'LOW' (light grey), and 'CUSTOM' (grey). Under 'REPLICATION FACTOR', the values are listed as 3, 3, 2, 1, and 2 respectively. In the 'Limits' section, the 'CLEANUP.POLICY' dropdown is set to 'delete'. To the right of the main interface is a detailed table of topic configurations:

Name	Value	IsDefault
compression.type	producer	true
leader.replication.throttled.replicas		true
message.downconversion.enable	true	true
min.insync.replicas	2	false
segment.jitter.ms	0	true
cleanup.policy	delete	false
flush.ms	9223372036854775807	true
follower.replication.throttled.replicas		true
segment.bytes	1073741824	false
retention.hours	168	false
flush.messages	9223372036854775807	true

# Delete a Topic

- Select Delete Topic from the ACTIONS dropdown menu

The screenshot shows the Apache Kafka Metrics UI for the topic 'Topics / call\_failed'. At the top right, it says 'Cluster: Cluster 1'. Below the navigation tabs (Metrics, Data Explorer, Configs, Latency), there's a date range selector ('a day') and an 'Actions' dropdown menu, which has 'Delete Topic' highlighted with a red box. The main section displays 'Producers (1)' with three entries: P0, P1, and P2. Each producer entry includes a green hexagonal icon with a checkmark, the ID (e.g., 30, 29, 31), the partition (P0, P1, P2), message sizes (10 KB in, 6 KB out for P0; 10 KB in, 7 KB out for P1; 10 KB in, 7 KB out for P2), and two small teal boxes at the end. To the right, there's a 'Consumer Groups (1)' section with a 'LAG' table showing 'producer-consum...' with a value of 0.

# Chapter Topics

---

## Using Streams Messaging Manager (SMM)

- Streams Messaging Manager Overview
- Producers, Topics, and Consumers
- Data Explorer
- Brokers
- Topic Management
- **Hands-On Exercise: Managing Topics using the CLI**
- **Hands-On Exercise: Connecting Producers and Consumers from the Command Line**
- 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

---

## Using Streams Messaging Manager (SMM)

- Streams Messaging Manager Overview
- Producers, Topics, and Consumers
- Data Explorer
- Brokers
- Topic Management
- Hands-On Exercise: Managing Topics using the CLI
- **Hands-On Exercise: Connecting Producers and Consumers from the Command Line**
- 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

---

## Using Streams Messaging Manager (SMM)

- Streams Messaging Manager Overview
- Producers, Topics, and Consumers
- Data Explorer
- Brokers
- Topic Management
- Hands-On Exercise: Managing Topics using the CLI
- Hands-On Exercise: Connecting Producers and Consumers from the Command Line
- **Essential Points**

## Essential Points

---

- Filter on topics, brokers, producers, and consumers using the **Overview** page
- End-to-end data flow displays associated producers, topics (or brokers), and consumer groups
- The data explorer displays message data by partition
- Profile details displays detailed information about the selected entity such as throughput and lag
- Create and manage topics using the **Topics** page. Here you can also specify pre-defined or advanced settings

# Bibliography

---

The following offer more information on topics discussed in this chapter

- [Cloudera SMM Documentation: Monitoring Clusters](#)
- [Cloudera SMM Documentation: Creating a Kafka Topic](#)



# Kafka Java API Basics

---

Chapter 6

# Course Chapters

---

- Introduction
- Kafka Overview
- Deploying Apache Kafka
- Kafka Command Line Basics
- Using Streams Messaging Manager (SMM)
- **Kafka Java API Basics**
- Improving Availability through Replication
- Improving Application Scalability
- Improving Application Reliability
- Analyzing Kafka Clusters with SMM
- Monitoring Kafka
- Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka Cluster
- Conclusion

# Kafka Command Line Basics

---

**By the end of this chapter, you will be able to**

- **Discuss the three client Kafka Java APIs**
- **Use the Java API to manage topics**
- **Create Kafka producers and Kafka consumers from the Java API**

# Chapter Topics

---

## Kafka Java API Basics

- **Overview of Kafka's APIs**
- Topic Management from the Java API
- Optional 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
- 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**
- Optional 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
- 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
- **Optional 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
- Essential Points

## Optional 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
- Optional 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
- 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`

# Chapter Topics

---

## Kafka Java API Basics

- Overview of Kafka's APIs
- Topic Management from the Java API
- Optional 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**
- 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
- Optional 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
- **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 Documentation](#)
- [Apache Kafka Java API Documentation](#)



# Improving Availability through Replication

---

Chapter 7

# Course Chapters

---

- Introduction
- Kafka Overview
- Deploying Apache Kafka
- Kafka Command Line Basics
- Using Streams Messaging Manager (SMM)
- Kafka Java API Basics
- **Improving Availability through Replication**
- Improving Application Scalability
- Improving Application Reliability
- Analyzing Kafka Clusters with SMM
- Monitoring Kafka
- Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka Cluster
- Conclusion

# Replication

---

**By the end of this chapter, you will be able to**

- **Understand Kafka replication**
- **Describe the assignment of partitions**
- **Discuss the impact of broker failure**
- **Configure preferred leaders**
- **Discuss replication best practices**

# Chapter Topics

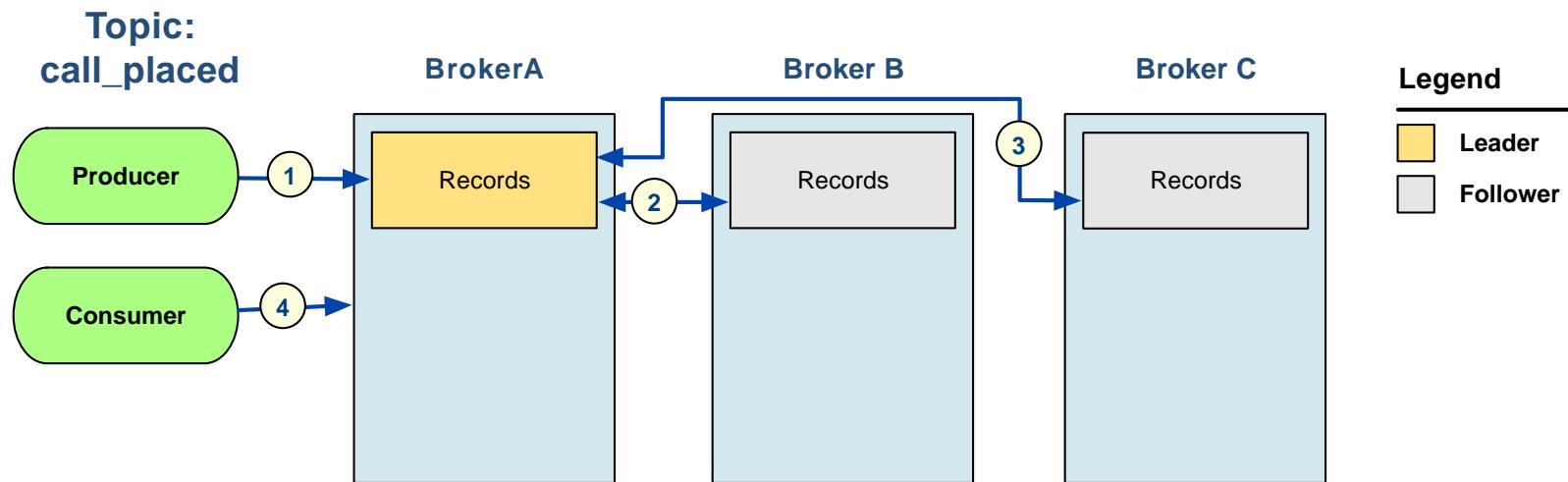
---

## Improving Availability through Replication

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

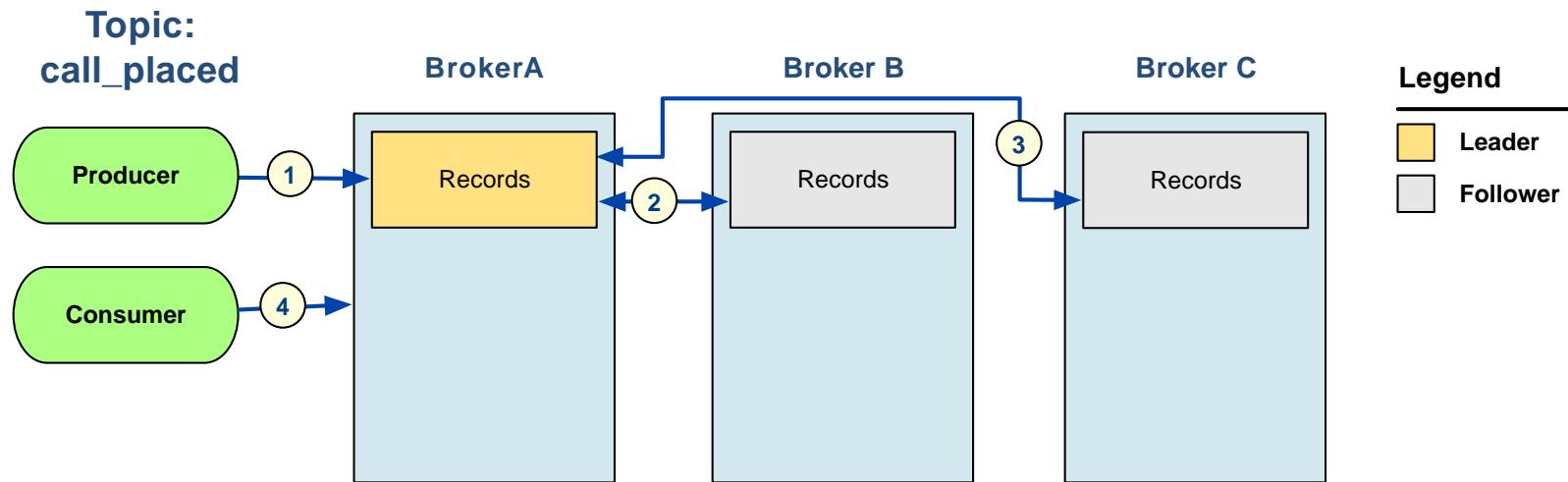
# Replication

- 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



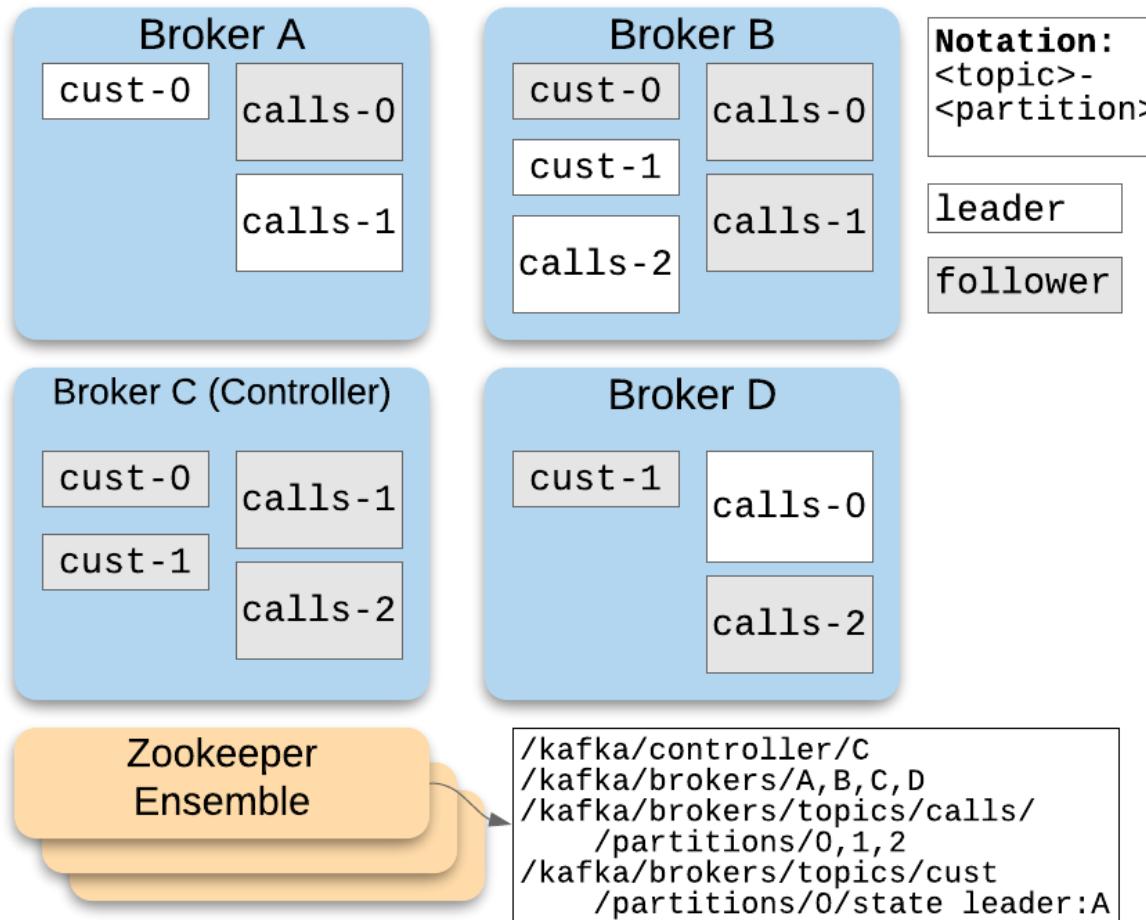
## Replication (Cont'd)

- 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



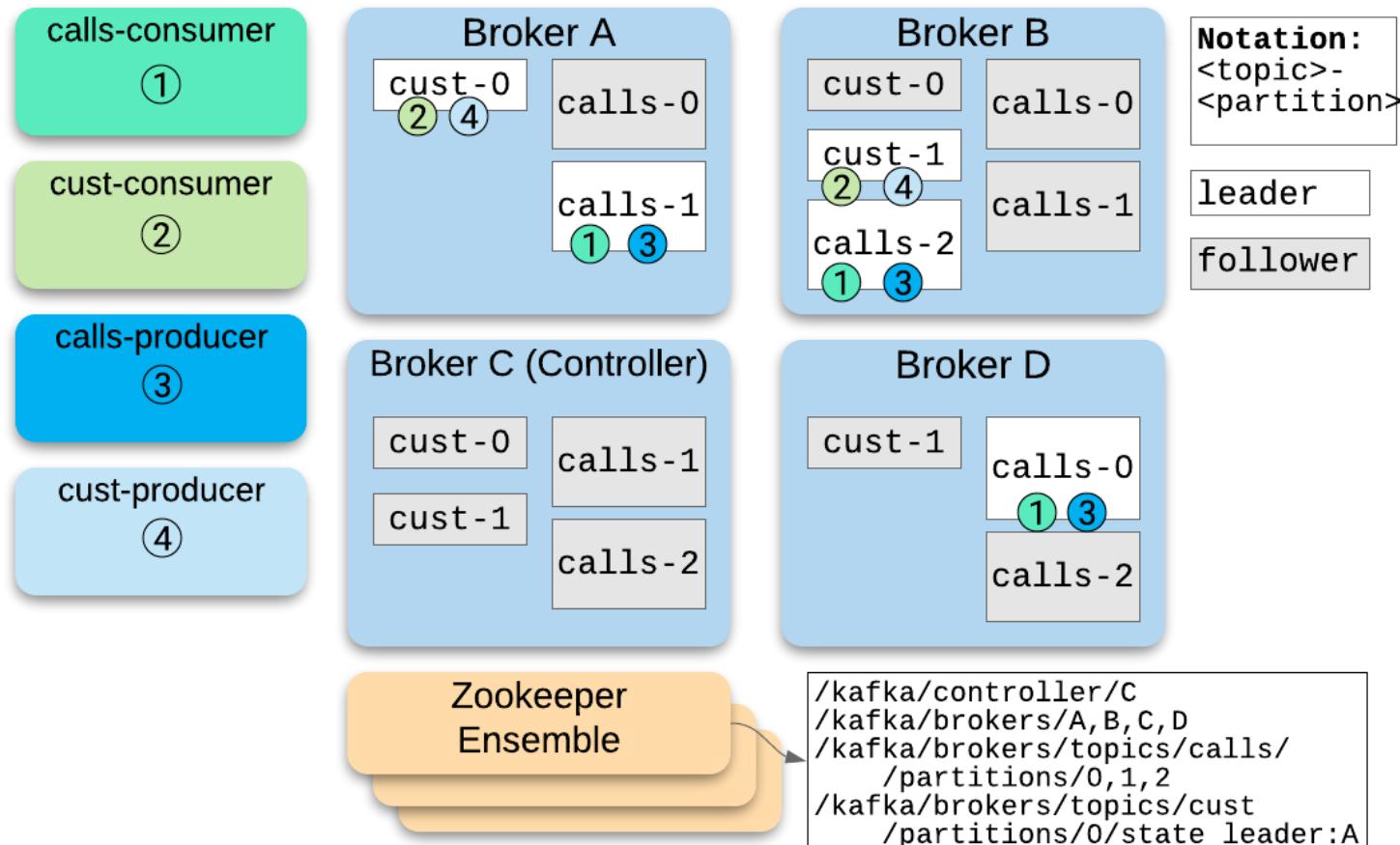
# Assignment of Partitions

- When a topic is created, Kafka controller assigns partition replicas to brokers
  - Partition assignments are stored in ZooKeeper



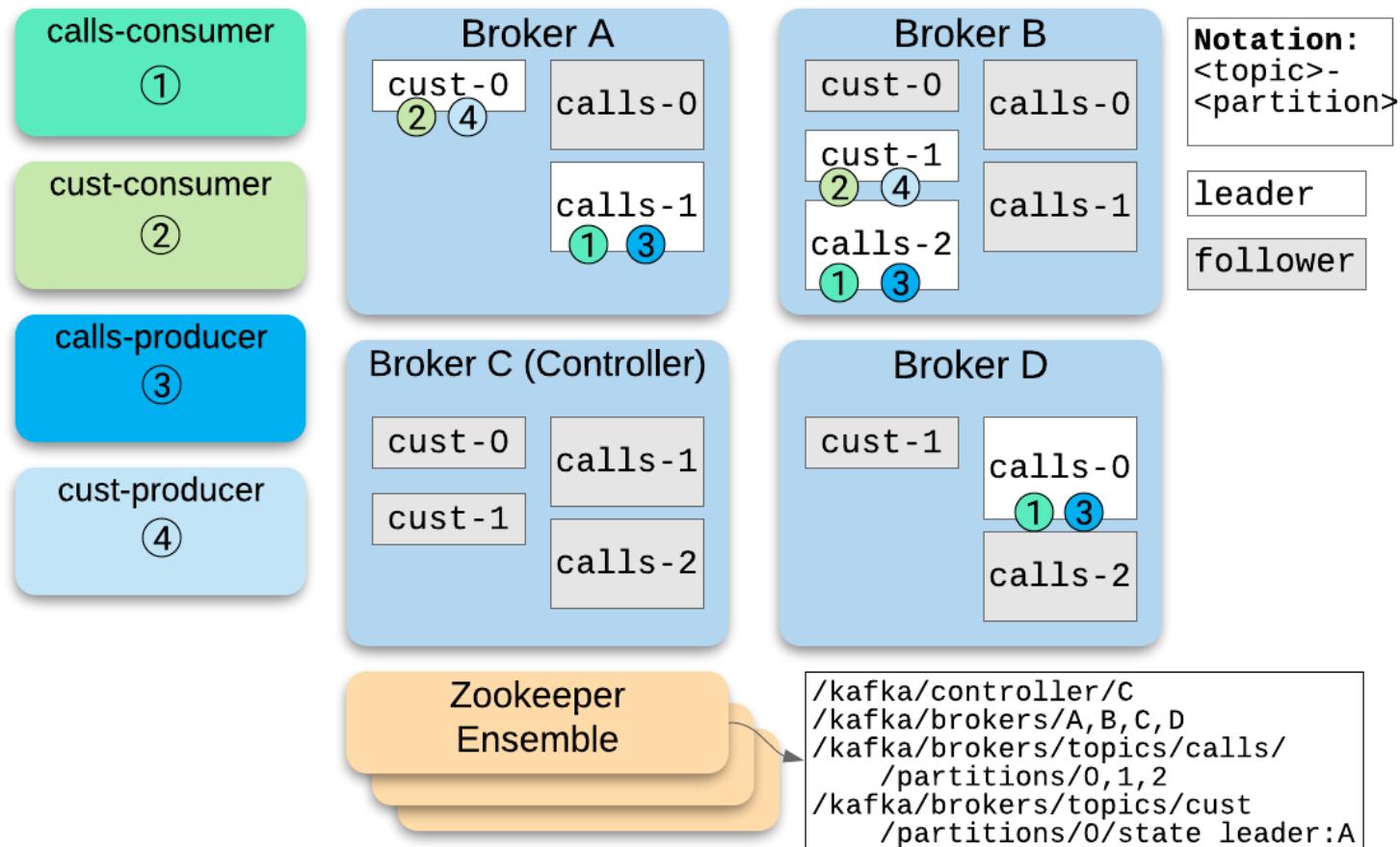
# Assignment of Partitions, Continued

- Brokers are assigned as either leader or follower for specific topic partitions
- Producers and Consumers send and receive messages from the leader



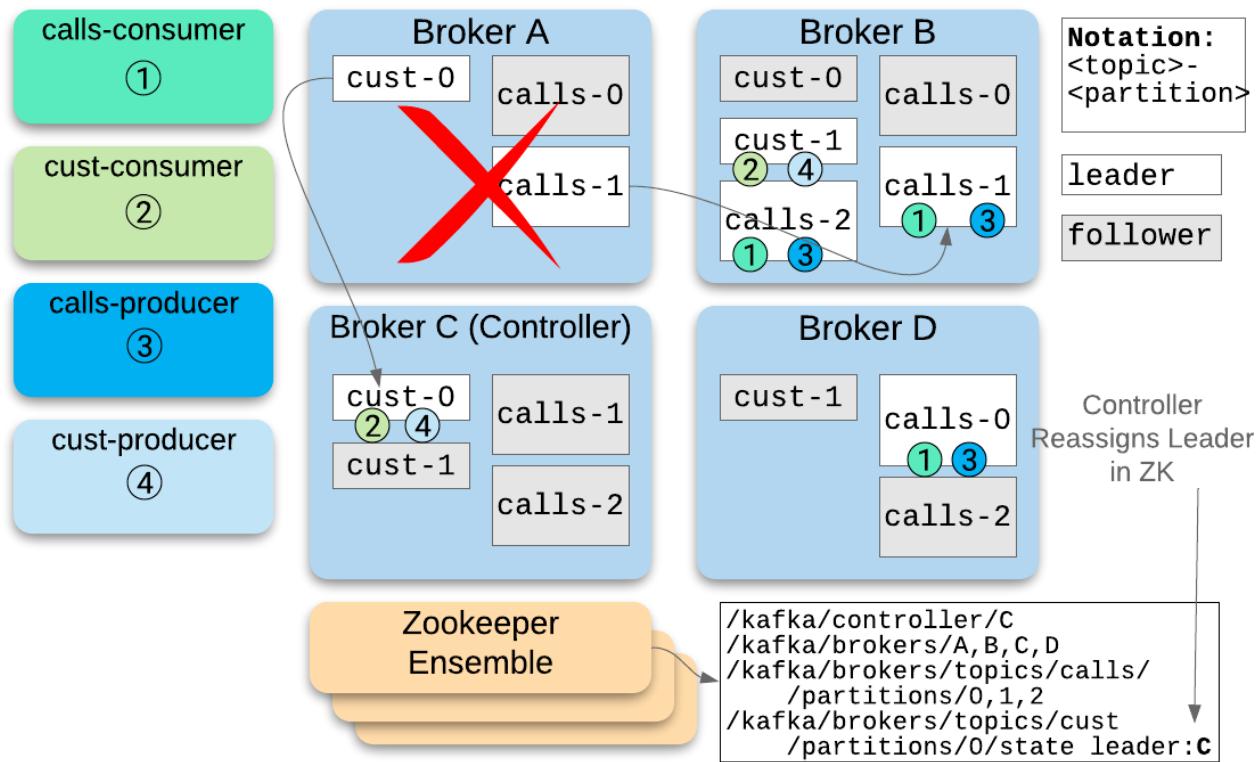
# Broker Failure

- Brokers must *heartbeat* to ZooKeeper every `zookeeper.session.timeout.ms` (default 6 seconds)
- Kafka controller watches ZooKeeper for broker heartbeats



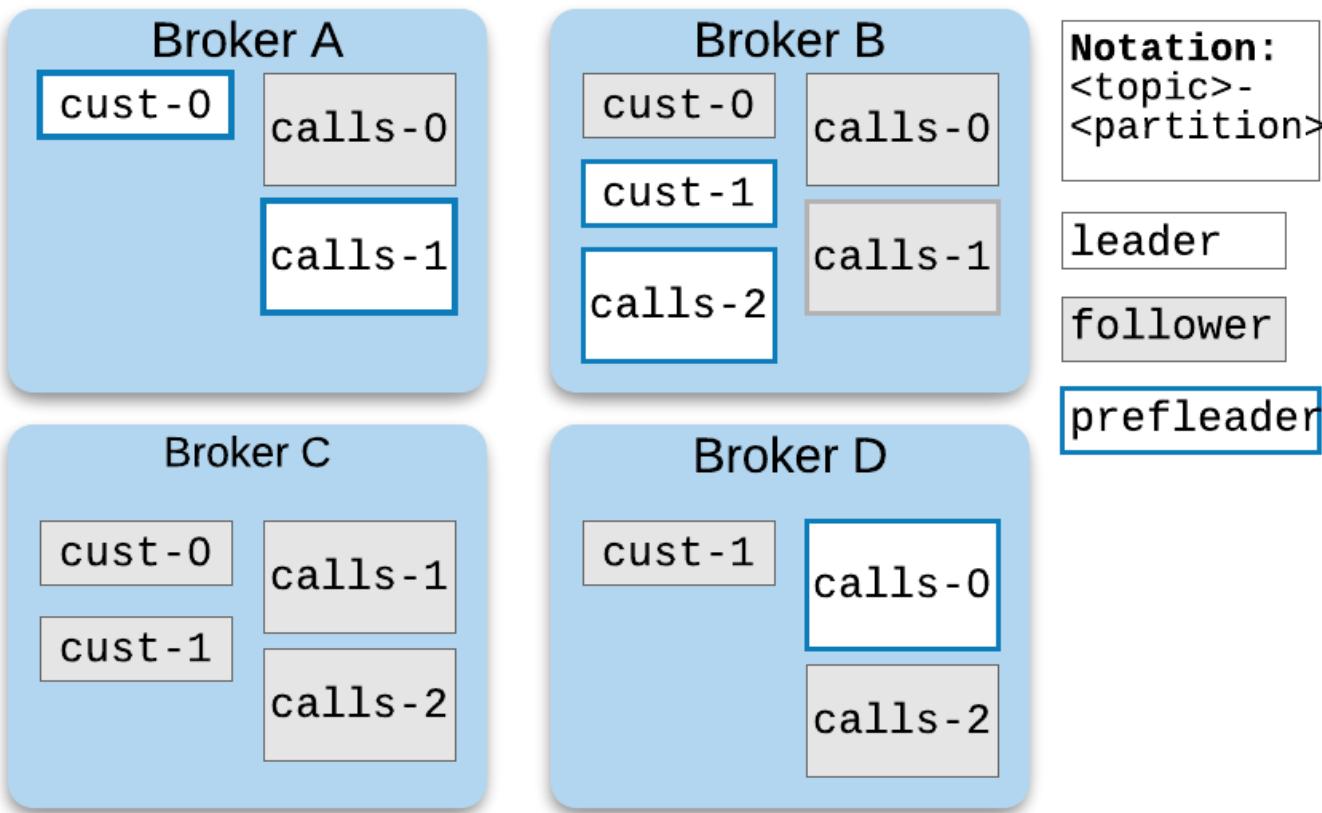
# Broker Failure, Continued

- In case of a "dead" broker, Kafka controller reassigned leadership of partitions
  - Producers and consumers learn of new leader through periodic *metadata requests*



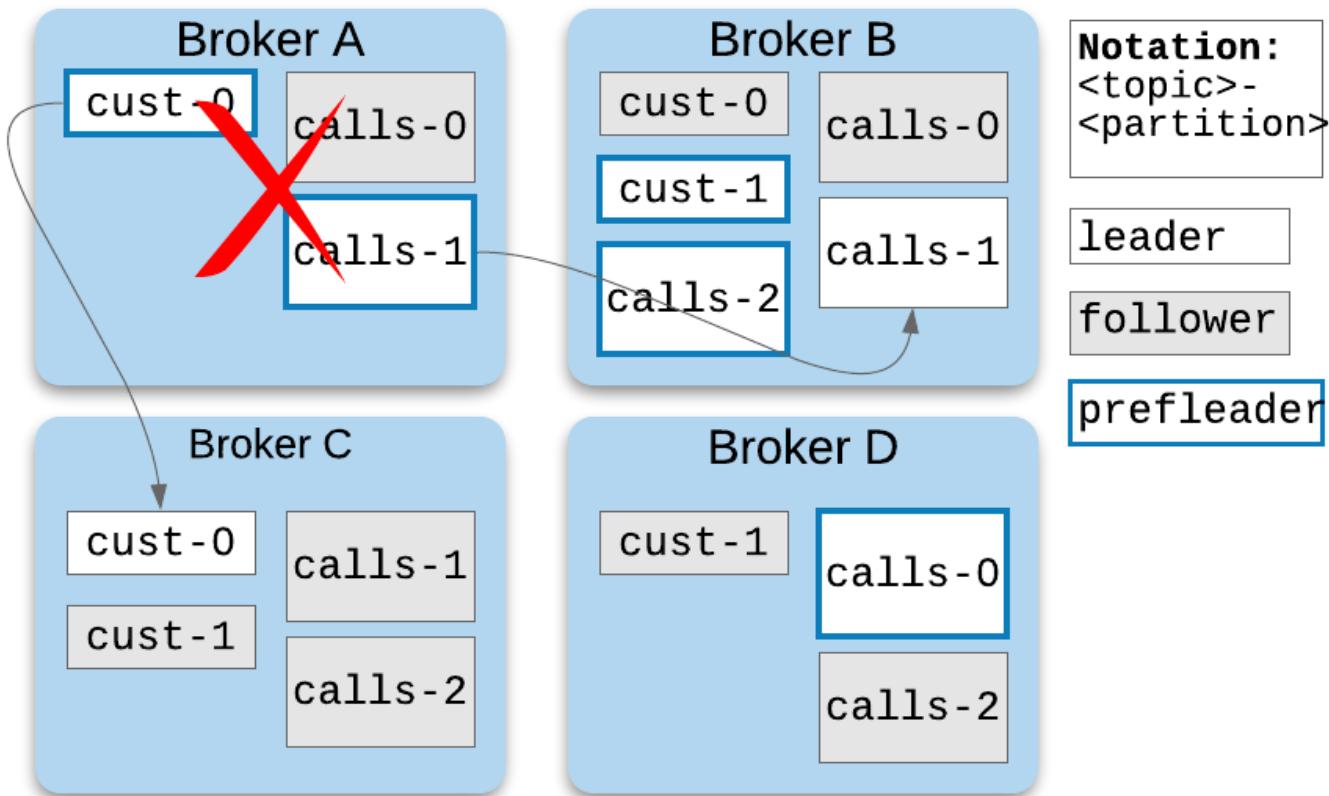
# Preferred Leaders

- When topics are created, the leader for each partition is designated as *preferred leader*
  - It's assumed that partition distribution is optimal when topics are created



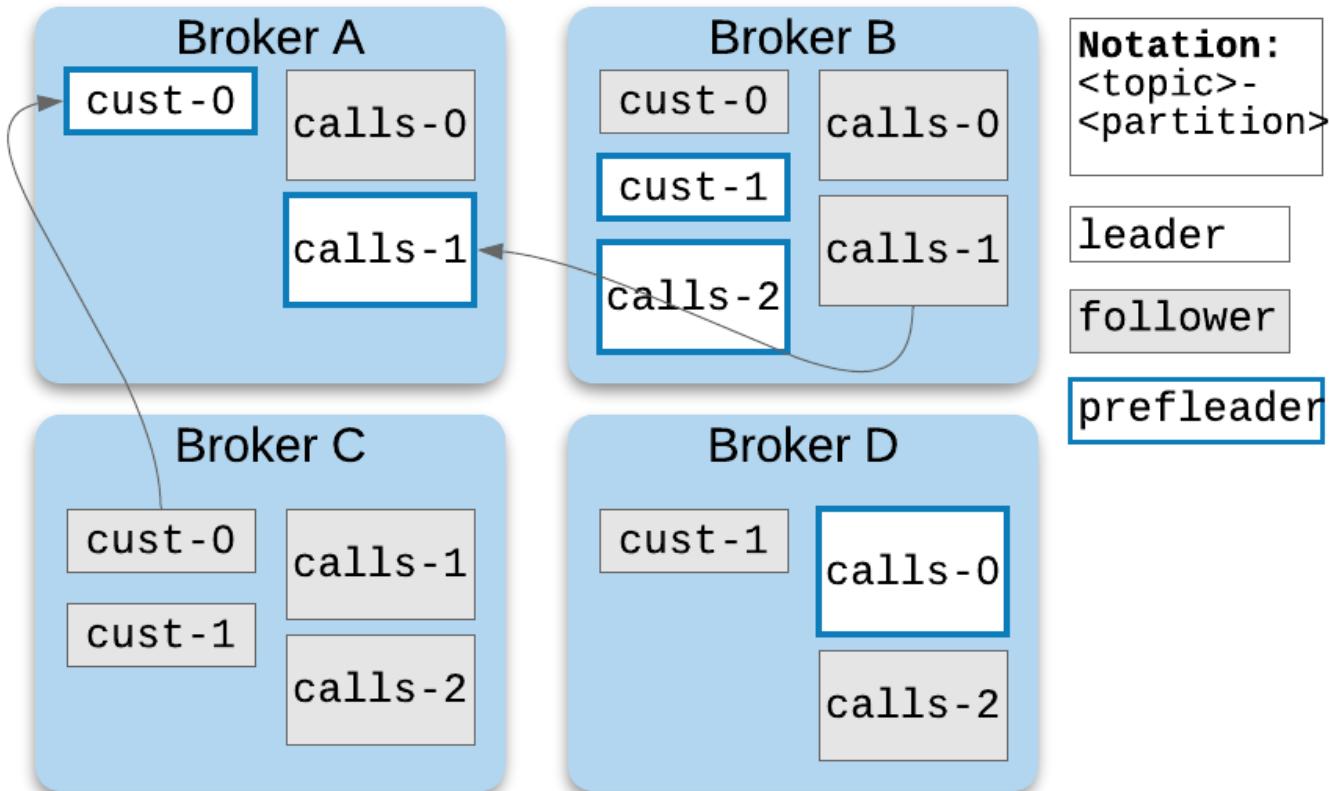
## Preferred Leaders, Continued

- When a broker fails, partition leader is assigned to another broker



## Preferred Leaders, Continued

- If the preferred leader returns it acts as a follower, until it catches up. It is then assigned as leader



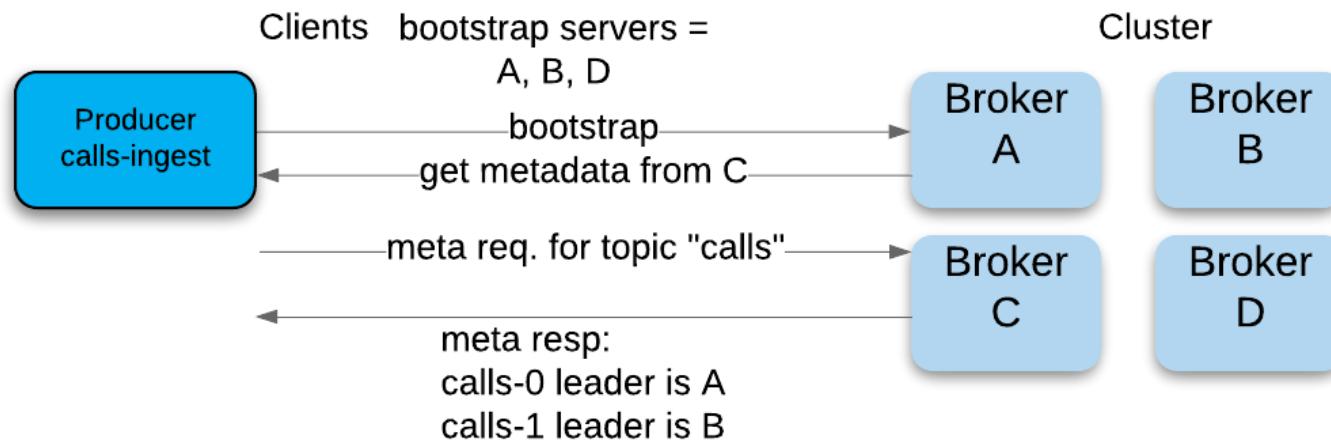
# Preferred Leader: Configuration and Inspection

---

- **auto.leader.rebalance.enable property (cluster-wide)**
  - Kafka controller periodically searches for partitions whose leader is not the preferred leader
  - If preferred leader is available and *in-sync*, leadership is assigned to preferred leader
- **Preferred leader election may be triggered manually**
  - See "Kafka Command Summary" exercise, section "Elect New Leader for Topic/Partitions"
- **kafka-topics --describe command will display preferred replica for each partition**
  - See "Kafka Command Summary" exercise, section "Viewing Partition Assignments for a Topic"
  - Hint: Preferred leader is denoted in an obscure way

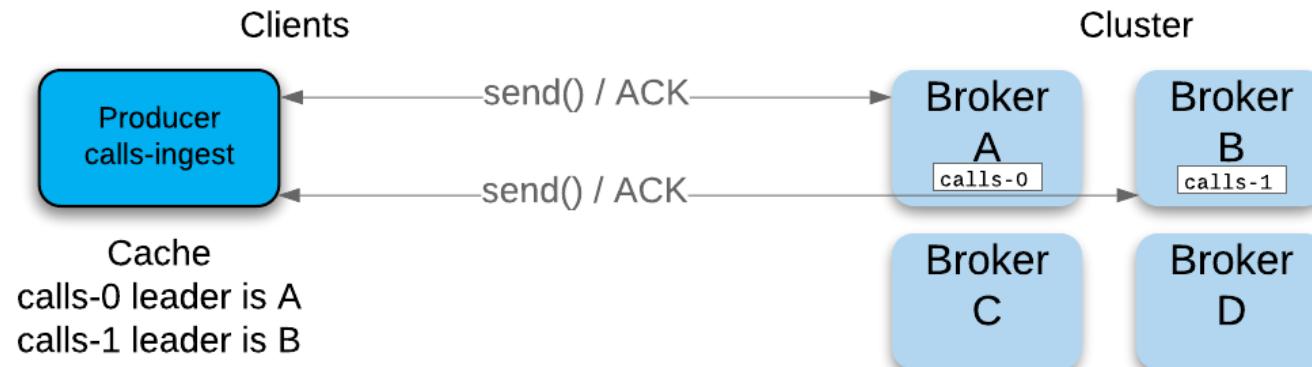
# Clients: Connection Overview

- **bootstrap-server** is usually a list of broker hostnames/ports which clients provide
  - May be actual hostnames and ports or virtual hostnames (e.g. load balancer(s))
  - Provides initial connection to Kafka cluster, including metadata requests
  - Clients cycle through the list



## Clients: Connection Overview, Continued

- Clients will cache broker and topic metadata
- Metadata is refreshed automatically at interval defined by `metadata.max.age.ms`
- Metadata is also refreshed if leader for a topic+partition is not available



## Metadata Bootstrap Messages: Normal Bootstrap

- Client (producer/consumer) DEBUG level logs:

```
DEBUG clients.NetworkClient: [Producer clientId=CL1]
Initialize connection to node worker3.example.com:9092 (id: -3
rack: null) for sending metadata request
DEBUG clients.NetworkClient: [Producer clientId=CL1] Sending
metadata request(type=MetadataRequest,
topics=hot_topic, allowAutoCreate=true) to node
worker1.example.com:9092 (id: -1 rack: null)
nodes = [worker2.example.com:9092 (id: 50 rack: null),
worker1.example.com:9092 (id: 51 rack: null) ...
Partition(topic = hot_topic, partition = 1, leader = 51,
replicas = [51,52,50], isr = [50,51,52], offlineReplicas =
[]),
Partition(topic = hot_topic, partition = 2, leader = 52,
replicas = [52,50,51], isr = [51,50,52], offlineReplicas =
[])...
controller = worker1.example.com:9092 (id: 51 rack: null))}
```

# Example of Metadata Refresh Caused By Leader Reassignment

---

- Client (producer) logs:

```
Got error produce response with correlation id 1496 on topic-partition hot_topic-1, retrying (2147483645 attempts left).
```

```
Error: NOT_LEADER_FOR_PARTITION
```

```
WARN internals.Sender: [Producer clientId=foo] Received invalid metadata error in produce request on partition hot_topic-1 due to
```

```
org.apache.kafka.common.errors.NotLeaderForPartitionException:  
This server is not the leader for that  
topic-partition... Going to request metadata update now
```

## Example of Metadata Refresh After Broker Crash

---

```
WARN clients.NetworkClient: [Producer clientId=CL1] Connection  
to node 52 (worker3.example.com/10.0.0.13:9092) could not be  
established. Broker may not be available.  
DEBUG clients.Metadata: Updating last seen epoch from 10 to 10  
for partition hot_topic-0  
DEBUG clients.Metadata: Updated cluster metadata version 3 to  
MetadataCache  
    nodes = [worker2.example.com:9092 (id: 50 rack: null),  
worker1.example.com:9092 (id: 51 rack: null)],
```

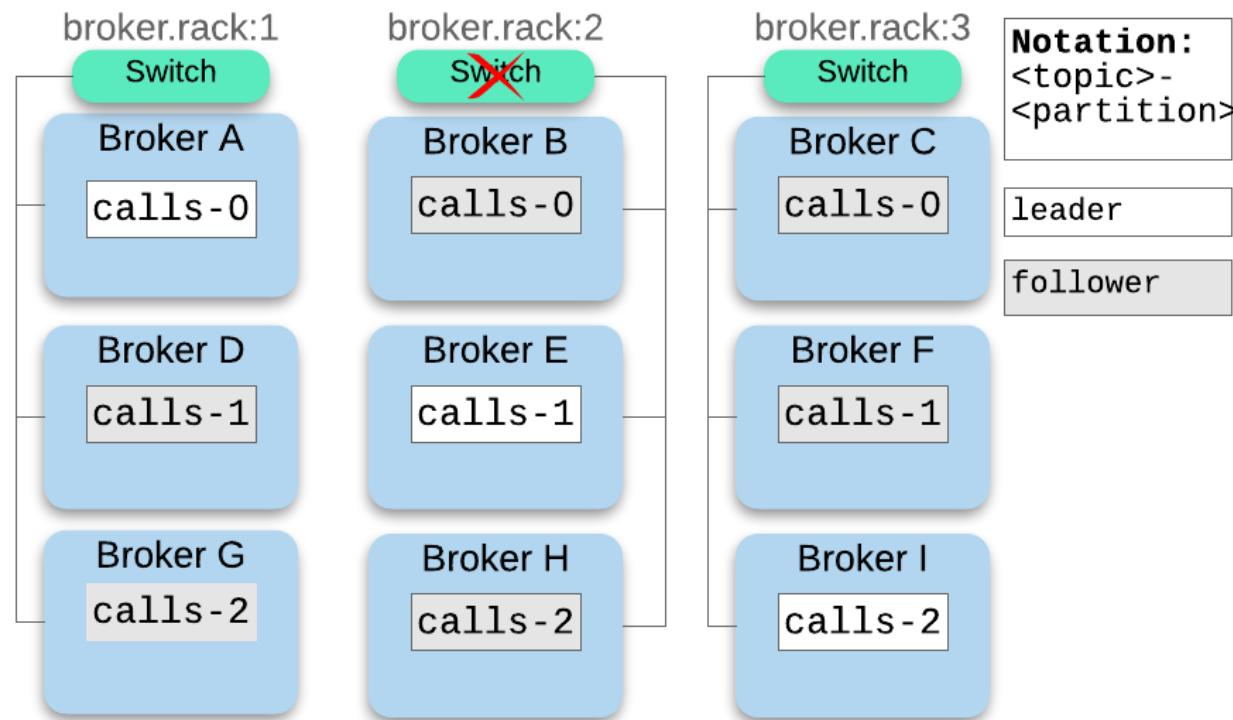
## Metadata / Bootstrap Summary

---

- **Metadata refreshes are common in clients**
  - `metadata.max.age.ms` is 5 minutes by default
- **Frequent metadata refreshes in clients may be an indicator of problems**
  - Example: Partitions being reassigned due to broker instability
- **Reference: "[Partitioning and Bootstrapping](#)" section at [kafka.apache.org](#)**

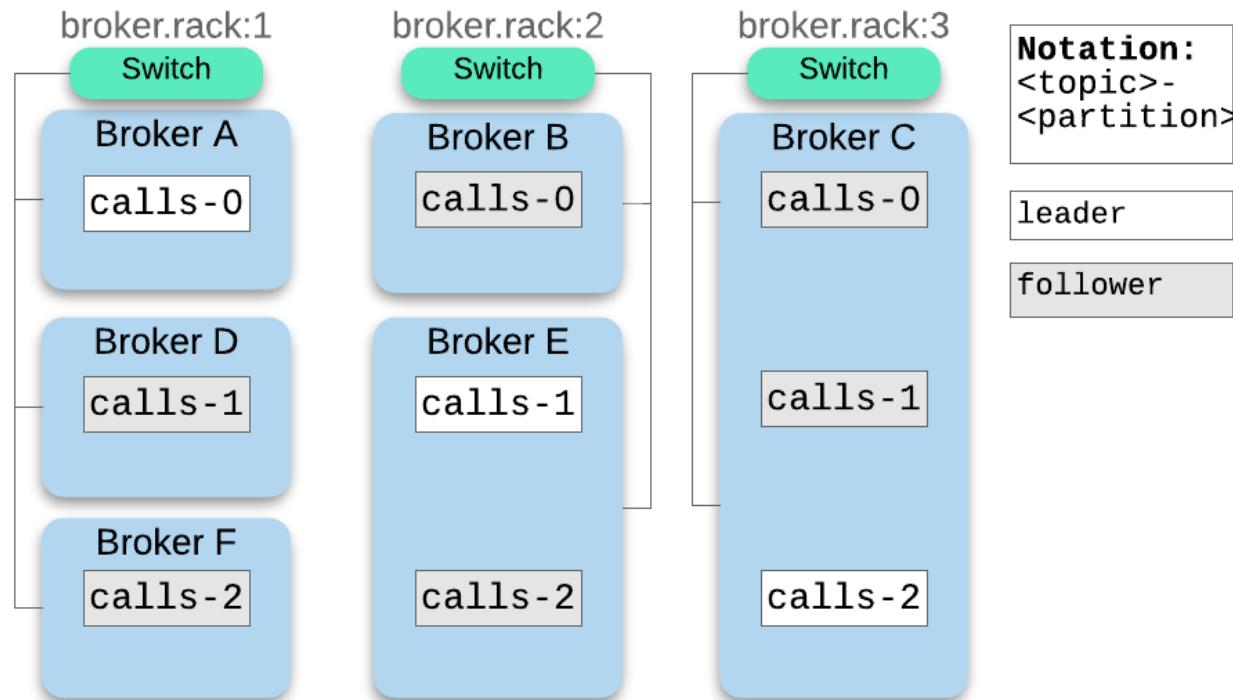
# Brokers: Rack Assignment

- Brokers may be assigned a `broker.rack`
  - A partition is assigned to brokers belonging to different racks (e.g. switches)
  - A single rack/switch failure will not affect multiple replicas of the same partition



# Rack Assignment: Traps

- Uneven assignment of brokers to racks results in uneven partition assignment



# Rack Assignment using Cloudera Manager

---

- **Specify `broker.rack=<rack>` property for each Broker using CM**
  - For each broker, modify "Kafka Broker Advanced Configuration Snippet (Safety Valve) for `kafka.properties`."

# Fault Tolerance Summary

---

- All brokers must heartbeat to ZooKeeper
  - If a leader does not heartbeat to ZooKeeper within a specified time (`zookeeper.session.timeout.ms`), 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 Summary (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://docs.cloudera.com/cdp-private-cloud-base/7.1.3/kafka-managing/topics/kafka-manage-migration-idmod.html>

‡ <https://docs.cloudera.com/cdp-private-cloud-base/7.1.3/kafka-managing/topics/kafka-manage-cli-reassign-overview.html>

# Chapter Topics

---

## Improving Availability through Replication

- Replication
- **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

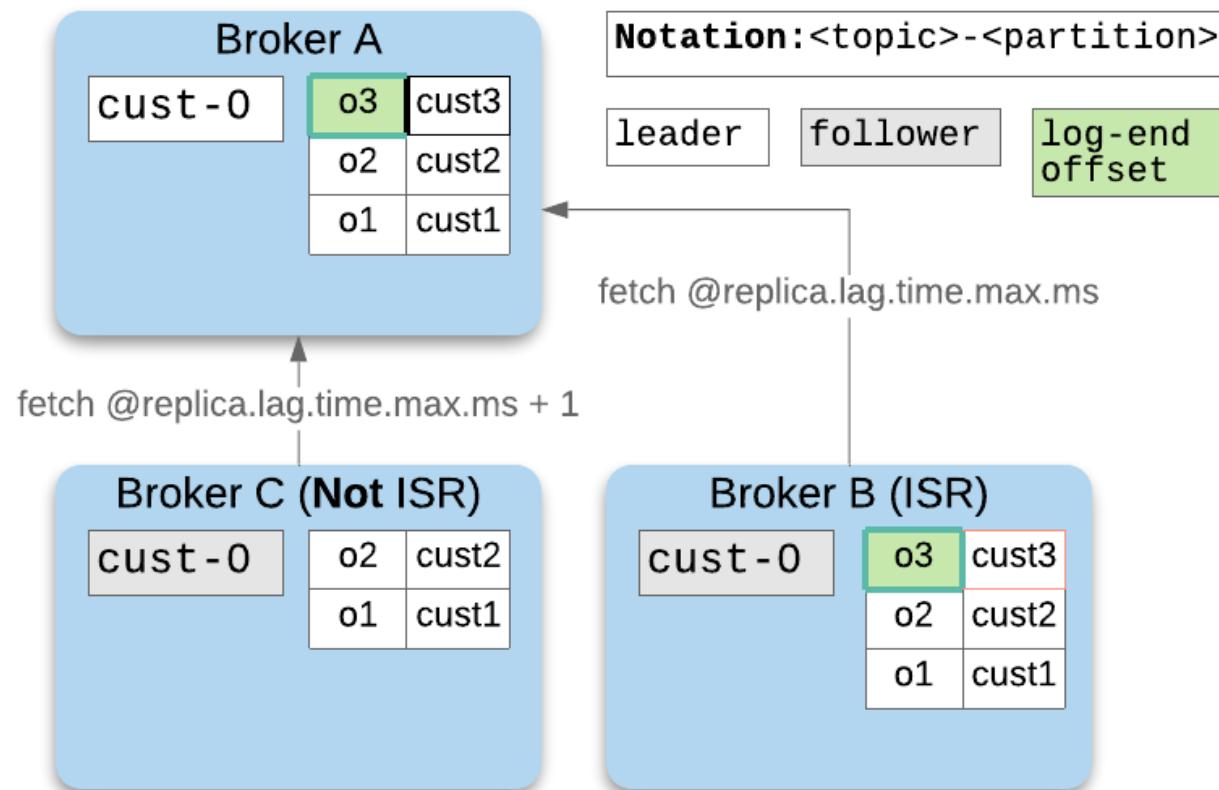
---

## Improving Availability through Replication

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

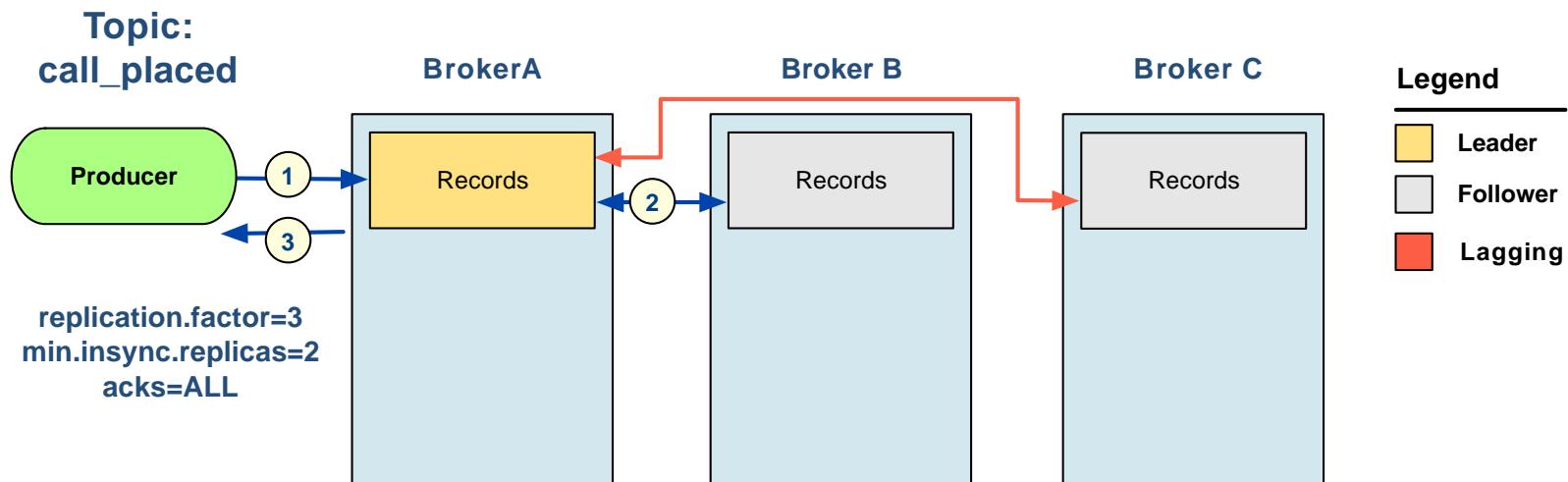
## In Sync Replicas

- A follower is considered "In Sync" if the following criteria is met:
  - A fetch request has been sent to the leader within `replica.lag.time.max.ms`
  - Follower is caught up to the *log end offset* of the partition



# 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`



## In Sync Replicas: Best Practices

---

- Brokers should remain in sync (should not be constantly going in and out of sync)
- If none of the followers are in sync and leader dies, partition will be unavailable (see following slides)
- Cloudera Manager alerts when brokers fall out of sync
- Out of sync replicas is normal in the case of broker restarts

# Out of Sync Replicas: Possible Causes

---

- **Soft network issues**
  - DNS caching / refreshes introducing latency periodically
- **Overloaded brokers**
  - Too many topics / partitions
  - High activity (throughput, requests from consumers / producers)
  - Check for long garbage collection (GC) pauses
- **Typical hardware issues (network, disk I/O, CPU)**
  - Low memory is not as common, but swapping is cause for investigation
  - Cloudera Manager alerts when swapping occurs

# Chapter Topics

---

## Improving Availability through Replication

- Replication
- 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
- 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 offers more information on topics discussed in this chapter

- [Balancing Apache Kafka Clusters Video](#)



# Improving Application Scalability

---

Chapter 8

# Course Chapters

---

- Introduction
- Kafka Overview
- Deploying Apache Kafka
- Kafka Command Line Basics
- Using Streams Messaging Manager (SMM)
- Kafka Java API Basics
- Improving Availability through Replication
- **Improving Application Scalability**
- Improving Application Reliability
- Analyzing Kafka Clusters with SMM
- Monitoring Kafka
- Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka Cluster
- Conclusion

# Scalability

---

**By the end of this chapter, you will be able to**

- **Describe how partitioning affects performance**
- **Implement consumer groups to impact performance**
- **Understand the causes for a consumer rebalance**

# Chapter Topics

---

## Improving Application Scalability

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

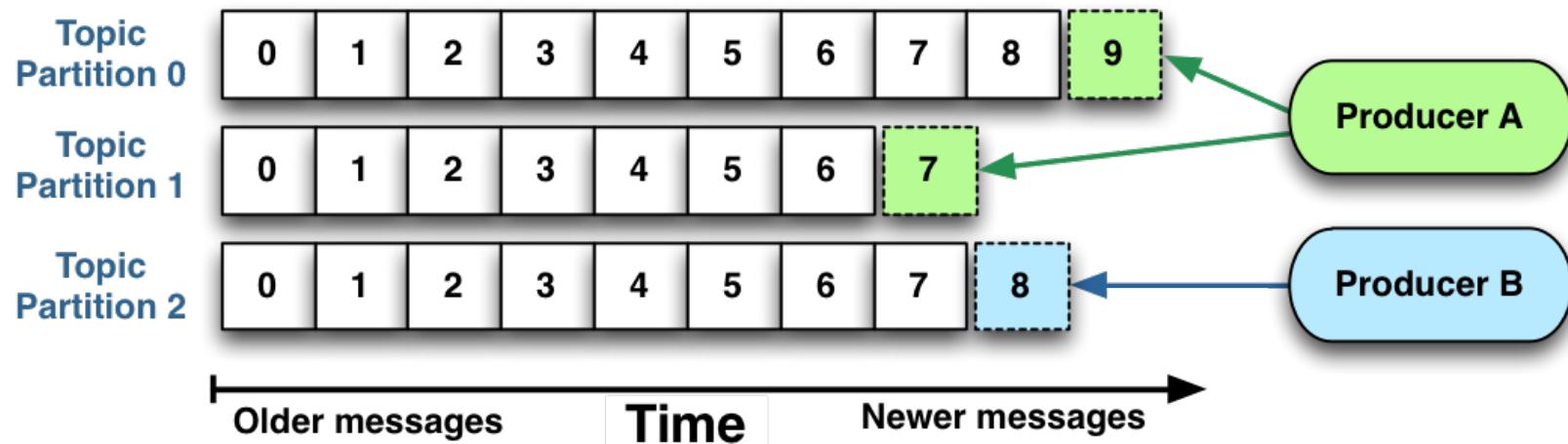
# Scaling Kafka

---

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

# How Messages Are Partitioned

---

- ***Producers specify a partition for each message***
- **Default behavior for producers based on the Java API (in order of priority):**
  1. Producers may explicitly set a partition for each message (not a common practice)
  2. If a key is specified for a message, a hash of the key is used to determine the partition
  3. If no key is specified for a message, a random partition is used for each batch of messages

# Partitioning Strategies

---

- Ordering of messages is guaranteed *per partition*
- For topics where ordering is important, the same message key may be used for related messages
- Example: `acct_txns` topic
  - Use the account\_id as the key of each message
  - Transactions with the same account\_id sent by the same producer are guaranteed to be in the order sent

Txn Date	Account ID/Key	Amount	Assigned Partition	Assigned Offset
12:00	1	100	0	0
12:00	2	100	1	0
12:00	3	100	1	1
12:00	4	100	0	1
12:01	2	-50	1	2
12:01	1	-10	0	2

## Partitioning Strategies (Continued)

---

- **Manually specifying partitions for messages is not typical**
  - An anti-pattern would be using partitions for specific types of data
  - E.g. **severe** alerts are sent to partition #1 of **alerts** topic
  - Recommendation: Use specific *topics* instead
- **Using message keys to send related messages to the same partition is reasonable**
- **For most use cases, partitions for the same topic should have relatively the same activity (number of messages, throughput)**
- **Note: If using default partitioning w/message keys, adding partitions to a topic will cause partition assignments to change**
  - Default algorithm is to use `hash(key) modulo num_partitions_in_topic`
  - Example: Messages for `account_id` # 1 are sent to partition #0
  - Number of partitions is increased for topic
  - Messages for `account_id` # 1 may now be sent to a different partition

# Chapter Topics

---

## Improving Application Scalability

- Partitioning
- How Messages are Partitioned
- **Hands-On Exercise: Observing How Partitioning Affects Performance**
- Consumer Groups
- Hands-On Exercise: Implementing Consumer Groups
- Consumer Rebalancing
- 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

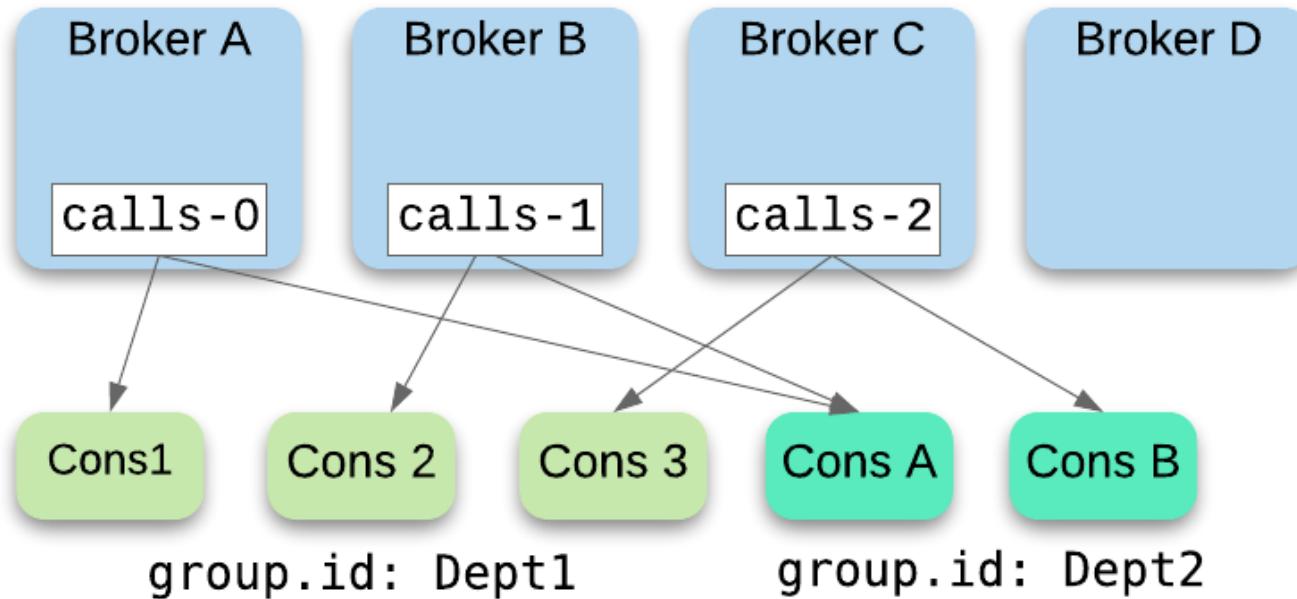
---

## Improving Application Scalability

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

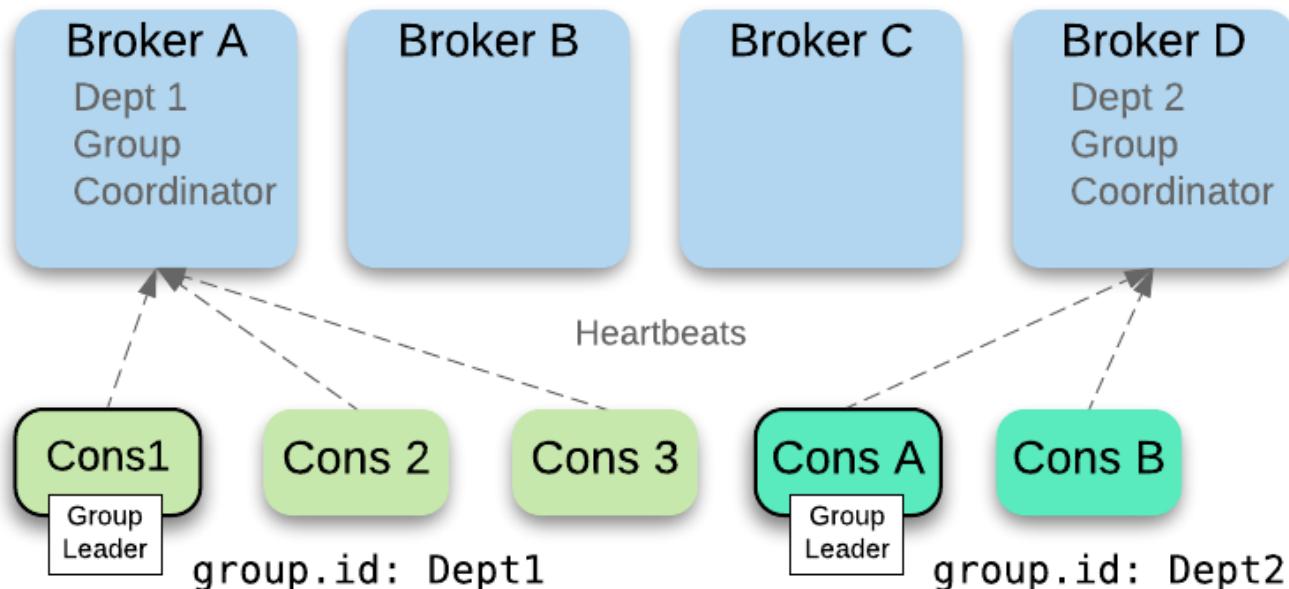
## Consumer Groups: Recap

---



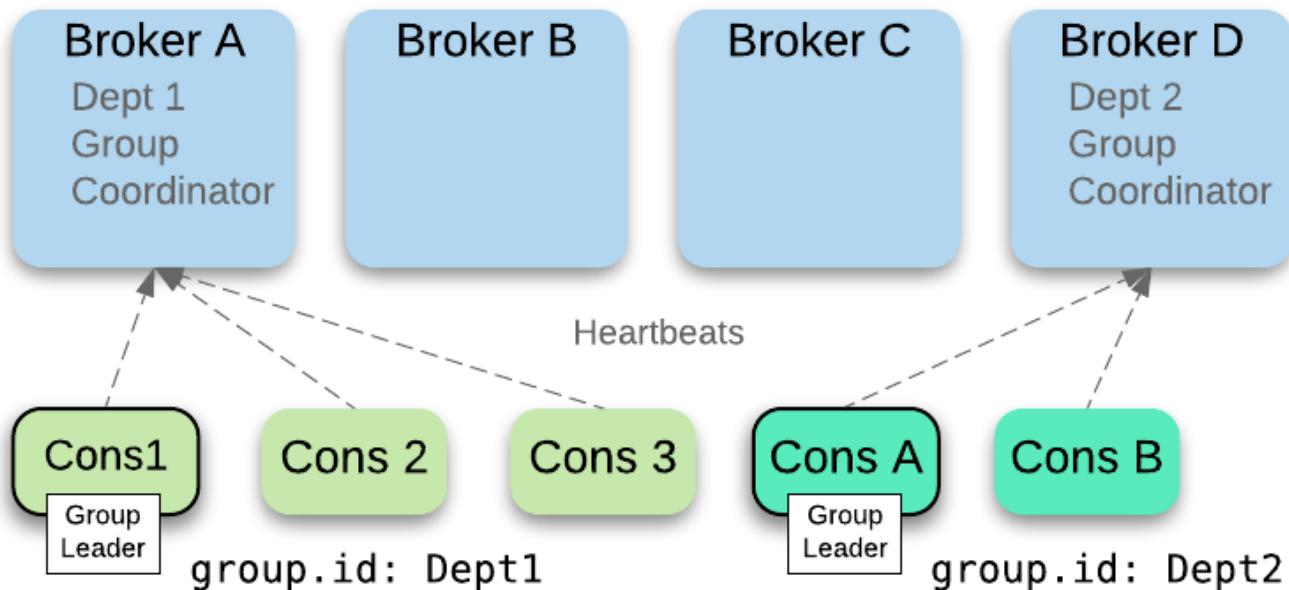
# Consumer Groups: Group Coordinator

- A (pseudo-random) broker is **Group Coordinator** for each consumer group
  - Elects a *Group Leader* among consumers
  - Monitors status of consumers using *heartbeats*
  - Informs Group Leader of new, lagging, dead consumers
  - Informs consumers of partition assignments / revocations
  - Performs offset commits on behalf of consumers



# Consumer Groups: Group Leader

- **Group Leader** is simply one of the consumers
  - Watches topics / patterns to which the group is subscribed to
  - Calculates partition assignments for group's consumers and informs the Group Coordinator of these assignments
- See [Kafka Client-Side Assignment Proposal](#)



# Chapter Topics

---

## Improving Application Scalability

- Partitioning
- How Messages are Partitioned
- Hands-On Exercise: Observing How Partitioning Affects Performance
- Consumer Groups
- **Hands-On Exercise: Implementing Consumer Groups**
- Consumer Rebalancing
- 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

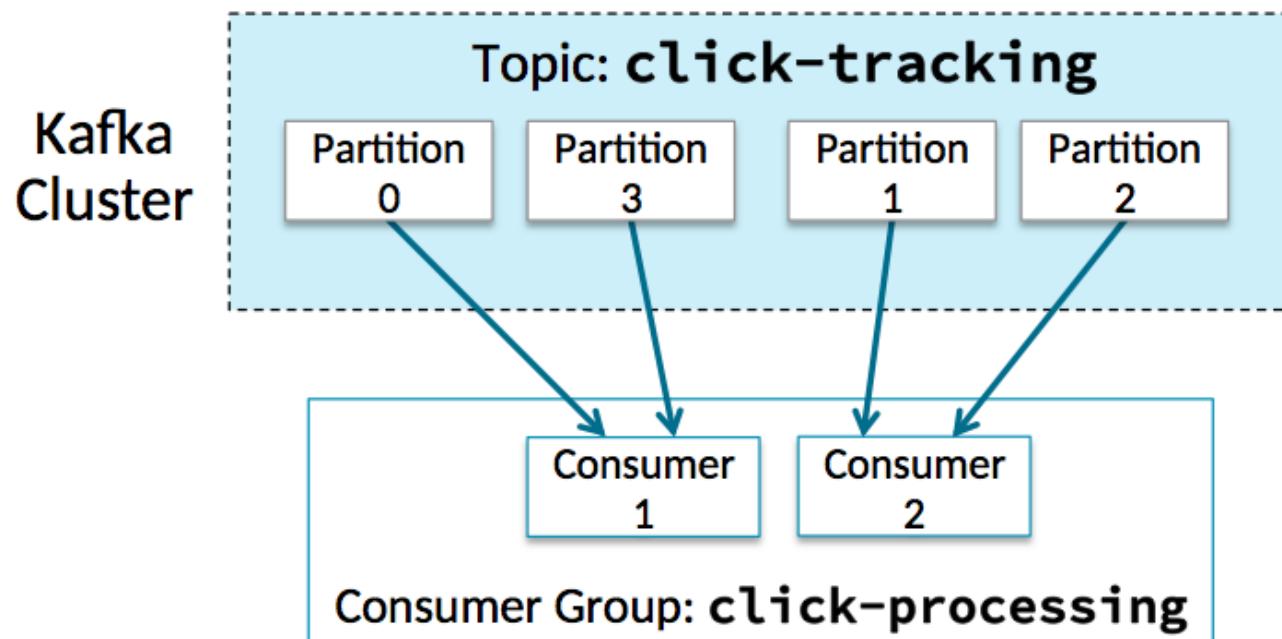
---

## Improving Application Scalability

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

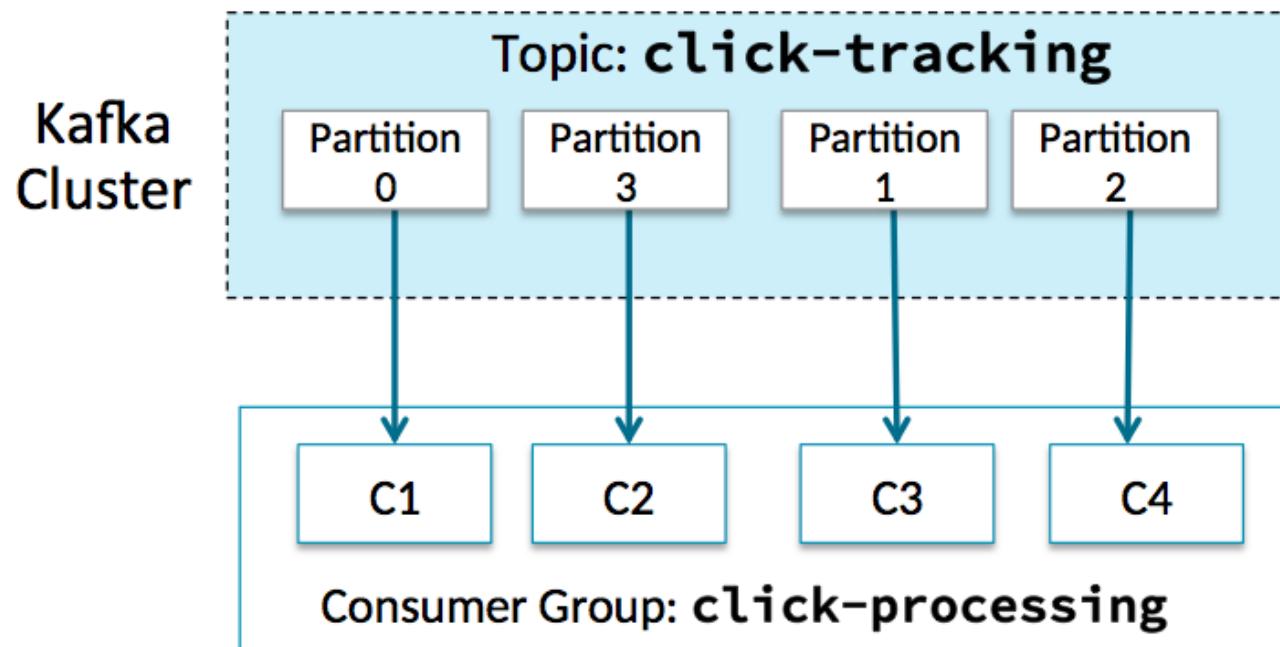
# Consumer Rebalancing

- Consumer group members are assigned partitions to consume
- Certain conditions can cause a *consumer rebalance*
- Partition assignments may change during a rebalance



## Consumer Rebalancing (Continued)

- In the example below, two new consumers have been added to the click-processing consumer group
- Partitions previously assigned to Consumer 1 and 2 have been reassigned to Consumers 3 and 4



# Causes for Consumer Rebalancing

---

- Possible causes for rebalancing include:
  - Addition and removal of consumers
  - Lagging consumers and consumer failure
  - Broker failures
  - Addition or removal of partitions or topics

# Chapter Topics

---

## Improving Application Scalability

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

## Essential Points

---

- **Producers specify a partition for each message**
  - Ordering of messages is guaranteed per partition
  - For topics where ordering is important, the same message key may be used for related messages
  - Manually specifying partitions for messages is not typical
- **The group leader calculates partition assignments for a group's consumers and informs the group coordinator**
- **Partition assignments may change during a consumer rebalance**

# Bibliography

---

The following offer more information on topics discussed in this chapter

- [Cloudera Blog: Scalability of Consumer Groups](#)
- [Consumer Groups and Fetching](#)



# Improving Application Reliability

---

Chapter 9

# Course Chapters

---

- Introduction
- Kafka Overview
- Deploying Apache Kafka
- Kafka Command Line Basics
- Using Streams Messaging Manager (SMM)
- Kafka Java API Basics
- Improving Availability through Replication
- Improving Application Scalability
- **Improving Application Reliability**
- Analyzing Kafka Clusters with SMM
- Monitoring Kafka
- Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka Cluster
- Conclusion

# Reliability

---

By the end of this chapter, you will be able to

- Manage message delivery types using producer and consumer options
- Discuss the steps during a logical transaction
- Analyze producer and consumer code that implements transactional messages
- Discuss how to handle consumer and producer failure

# Chapter Topics

---

## Improving Application Reliability

- **Delivery Semantics**
  - Optional Instructor-Led Demonstration: ISRs vs. ACKs
  - Producer Delivery
  - Hands-On Exercise: Idempotent Producer
  - Transactions
  - Hands-On Exercise: Transactional Producers and Consumers
  - Handling Consumer Failure
  - Offset Management
  - Hands-On Exercise: Detecting and Suppressing Duplicate Messages
  - Hands-On Exercise: Handling Invalid Records
  - Handling Producer Failure
  - Essential Points

# Producer Delivery Semantics

---

- Producers can be configured for three basic delivery types:
  - At most once delivery
  - At least once delivery
  - Exactly once delivery

# Producers: Best Practices

---

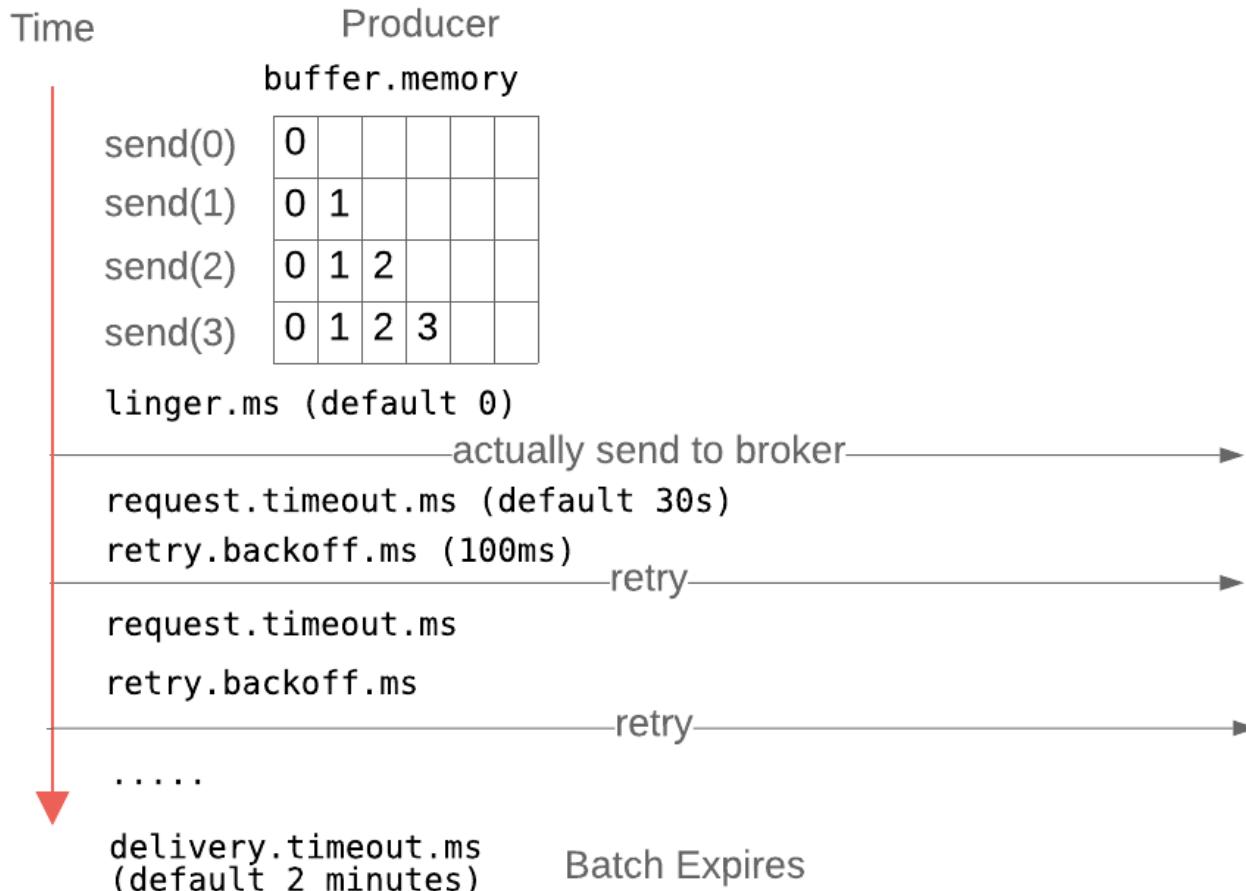
- **Kafka API provides reasonable guarantees for delivery**
- **Many responsibilities fall outside of Kafka API**
- **Questions to consider:**
  - How to monitor producer health?
  - How to track what's been sent? (e.g. rows from a database table)
  - What happens if a producer crashes? Restart?
  - Where do "new" producers resume?
  - What if producer was backlogged? Messages queued but never sent?

# Producer Options Affecting Delivery Types (1)

Option Name	Description	Default
acks	Number of brokers which must acknowledge receipt of a message.	1
retries	Number of times a producer will resend a message which failed due to a <a href="#">Retriable Exception</a>	2147483647
request.timeout.ms	Amount of time producer will wait for response from broker before retrying or failing a request	30_000
retry.backoff.ms	Milliseconds to wait if a request times out.	100
delivery.timeout.ms	Upper bound on the time to report success or failure after a call to send()	120_000
max.in.flight.requests.per.connection	Maximum "in flight" send() requests made to same leader broker. Out of order messages can occur if value is > 1	5

# Producer timeout settings

---



## Producer Options Affecting Delivery Types (2)

Option Name	Description	Default
<code>enable.idempotence</code>	Helps prevent brokers from saving duplicate messages sent by the same producer <i>instance</i> .	<code>false</code>
<code>transactional.id</code>	Unique ID (provided by developer!) to <i>each instance</i> of a producer. Used to facilitate atomic commits.	<code>null</code>
<code>transaction.timeout.ms</code>	Maximum amount of time a transaction will remain open. If this amount of time is exceeded, messages in this transaction will be aborted	<code>60_000 ms</code>

# Topic Options affecting Delivery Types

---

Option Name	Description	Default
<code>replication.factor</code>	Generally, topics requiring durability of messages should have a <code>replication.factor</code> of 3 or higher	<code>null</code>
<code>min.insync.replicas</code>	Minimum number of <i>in-sync</i> replicas which must acknowledge receipt of a message, including leader. <b>Note:</b> This takes effect when producers specify <code>acks=ALL</code>	<code>null</code>

# Consumer Options affecting Delivery Types

---

Option Name	Description	Default
<code>isolation.level</code>	Controls how consumers receive transactional messages (See later)	<code>read_uncommitted</code>

## Producer Delivery: At Most Once (Fire and Forget)

---

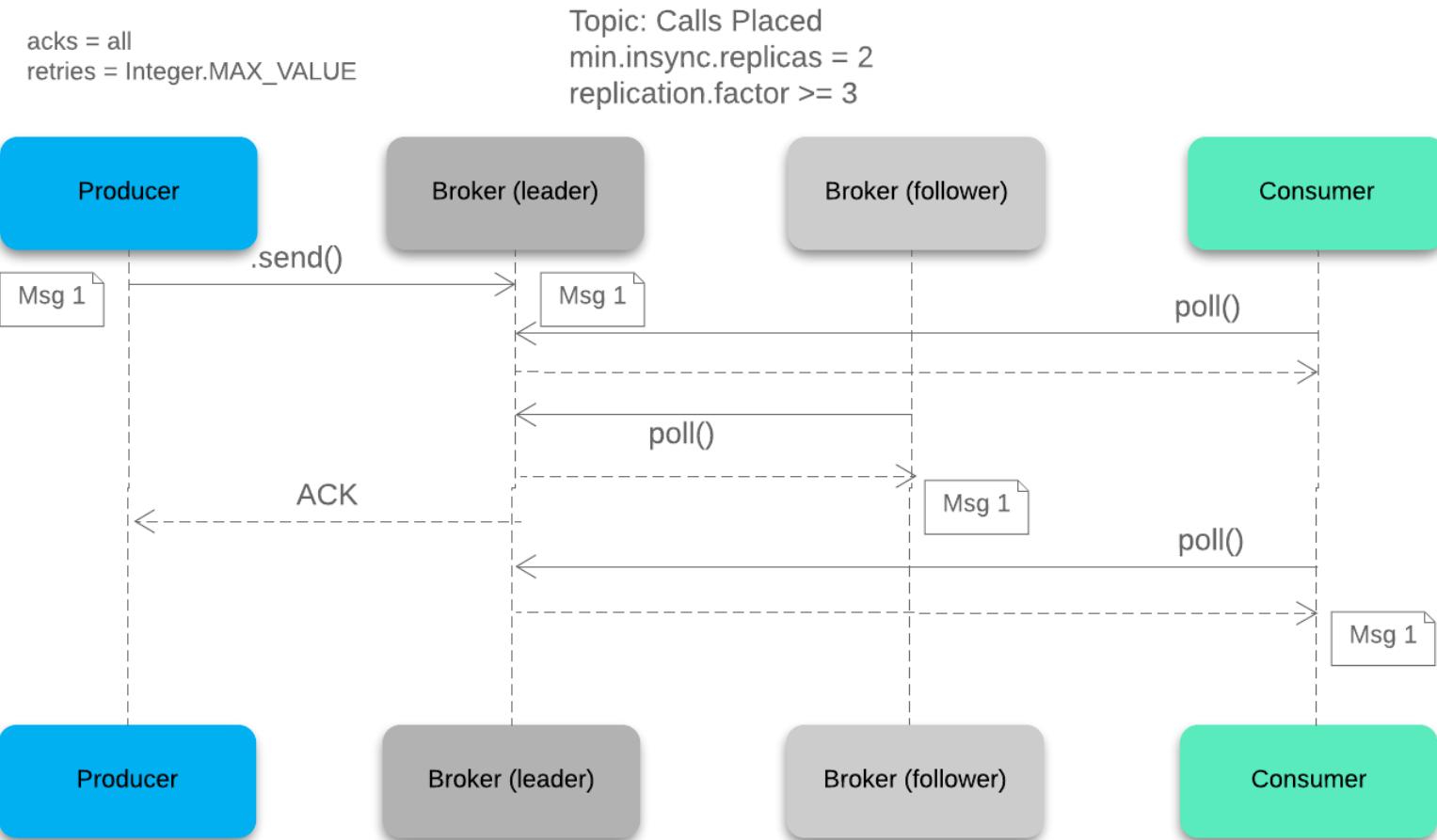
- Provides very fast delivery, but no guarantee of delivery
- To enable, set acks=0
  - retries is ignored by producers when acks=0
- Messages may be lost for many reasons (broker failure, network timeout, etc)

# Producer Delivery: At Least Once

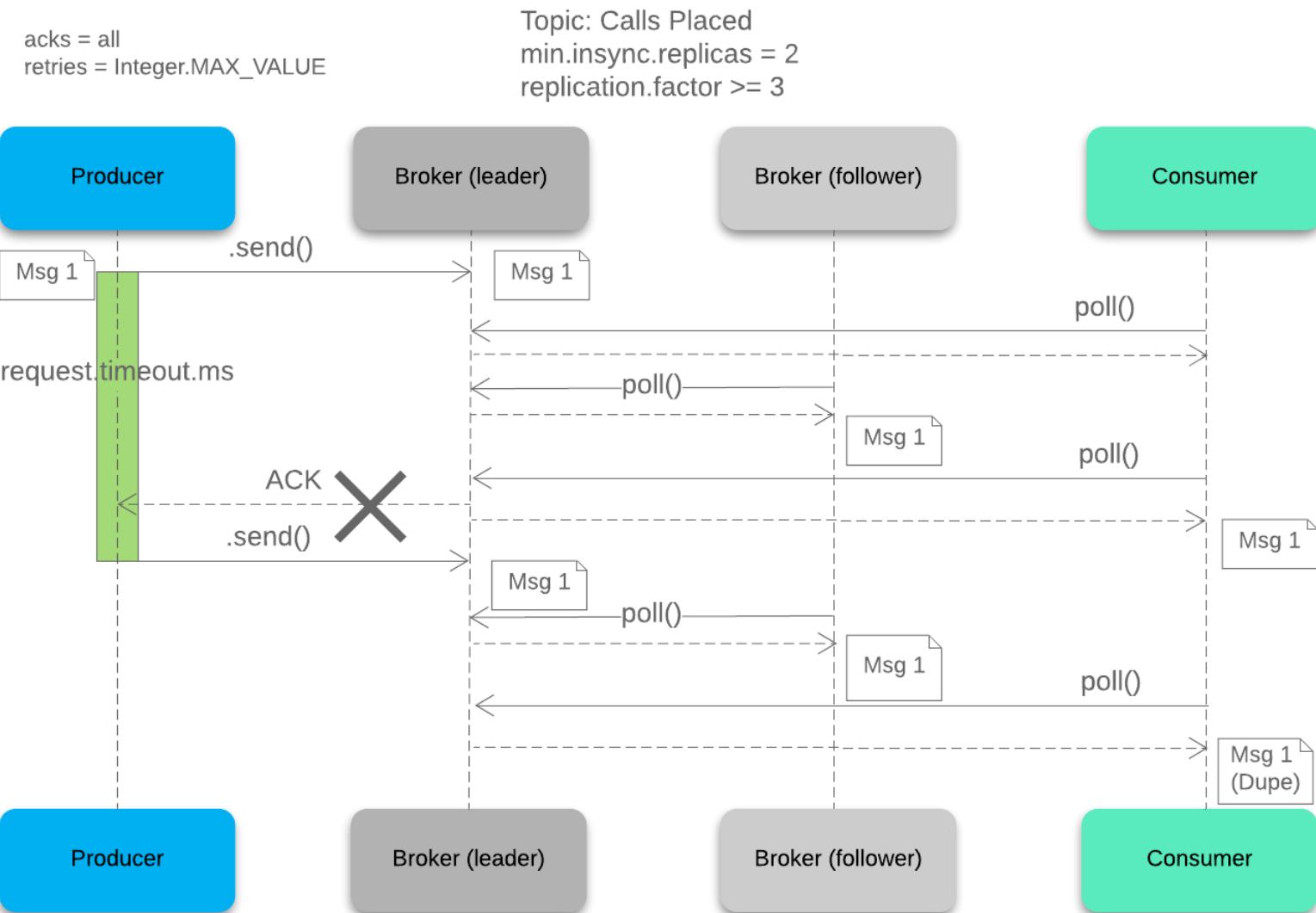
---

- **Provides guarantee of message delivery**
  - Duplicate messages may occur
  - Producer may not receive ack from broker within `delivery.timeout.ms`, and retry sending the message
- **Producer responsibilities:**
  - Require acknowledgements from more than one broker
  - Retry failed message deliveries
  - Developers must handle **producer crashes and restarts**
- **Topic responsibilities:**
  - Set replication factor to withstand broker failures
  - Set `min.insync.replicas` to ensure at least 2 brokers receive messages before acknowledgement sent

# Producer Delivery: At Least Once



# Producer Delivery: At Least Once (Duplicate Msg)



## Recommended Settings: At Least Once Delivery

---

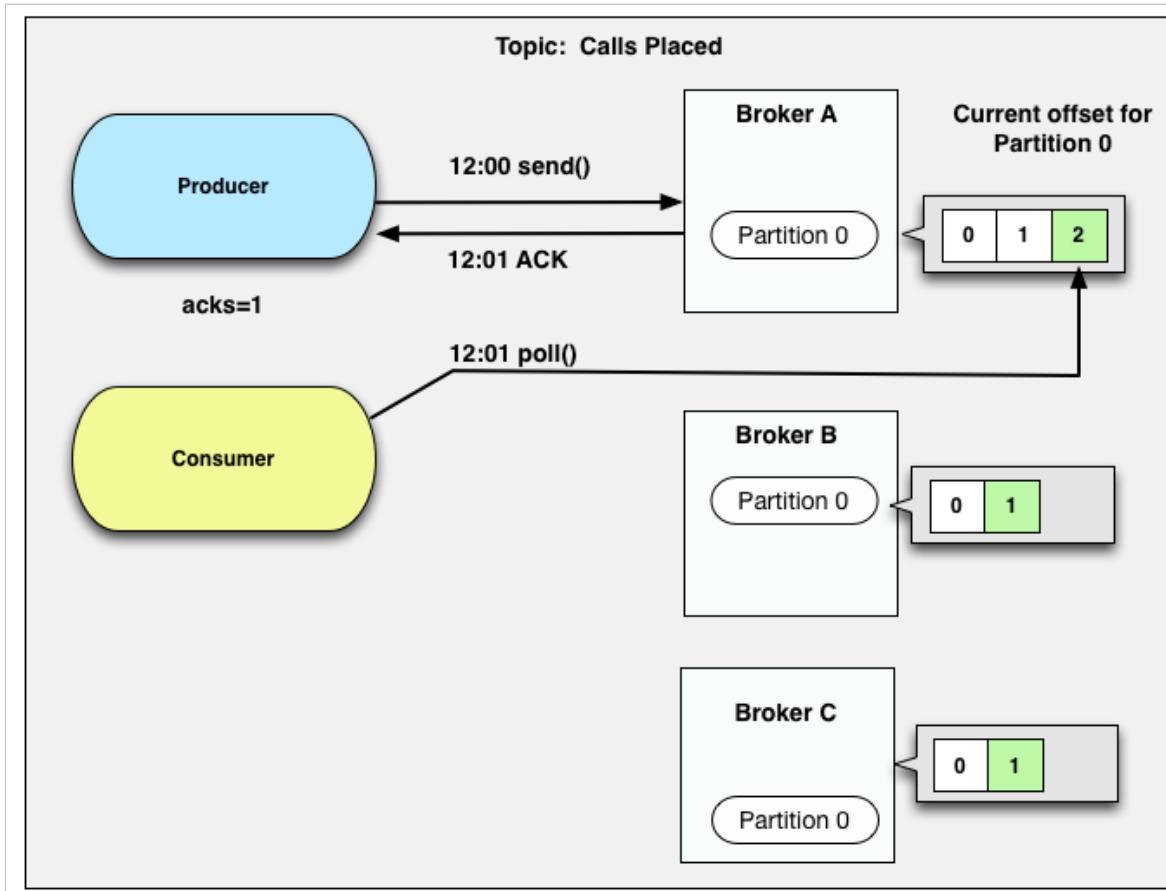
Option Name	Scope	Value
acks	Producer	ALL
retries	Producer	Long.MAX_VALUE
replication.factor	Topic	3
min.insync.replicas	Topic	2

## An Aside: ACKS and In Sync Replicas

---

- A follower (a.k.a replica) of a topic+partition is considered **in-sync** for a topic +partition if:
  - Follower has fetched messages from the leader within `replica.lag.time.max.ms`
  - Follower is "caught up" to leader's latest set of messages
- **acks** is the number of **in-sync** replicas which must receive a message
- **acks=ALL** requires acknowledgement from all **in-sync** replicas
  - **Problem:** If there's only one **in-sync** replica, ACK will be sent to producer after leader receives message
  - Messages may still be lost if leader acknowledges but crashes before followers receive message replica
  - **Solution:** Set `min.insync.replicas > 1` for topic
- **acks=1** requires an acknowledgement from the **leader broker only**
  - Messages may still be lost if leader acknowledges but crashes before followers receive message replica
  - **Solution:** Don't use `acks=1`

# ACKS = 1



- What if Broker A crashes at 12:02 below? Answer: Broker B or C take over and subsequent consumers do not see message 2

# Chapter Topics

---

## Improving Application Reliability

- Delivery Semantics
- **Optional Instructor-Led Demonstration: ISRs vs. ACKs**
- Producer Delivery
- Hands-On Exercise: Idempotent Producer
- Transactions
- Hands-On Exercise: Transactional Producers and Consumers
- Handling Consumer Failure
- Offset Management
- Hands-On Exercise: Detecting and Suppressing Duplicate Messages
- Hands-On Exercise: Handling Invalid Records
- Handling Producer Failure
- Essential Points

## Optional Instructor-Led Demonstration: ISRs vs. ACKs

---

- Your instructor will now demonstrate the comparison of ISRs and ACKs
  - In this demo, you will see the difference between producers using ACKS=1 versus ACKS=all
  - See "Comparing ISRs and ACKs" in the "Demos" section of the Exercise Guide

# Chapter Topics

---

## Improving Application Reliability

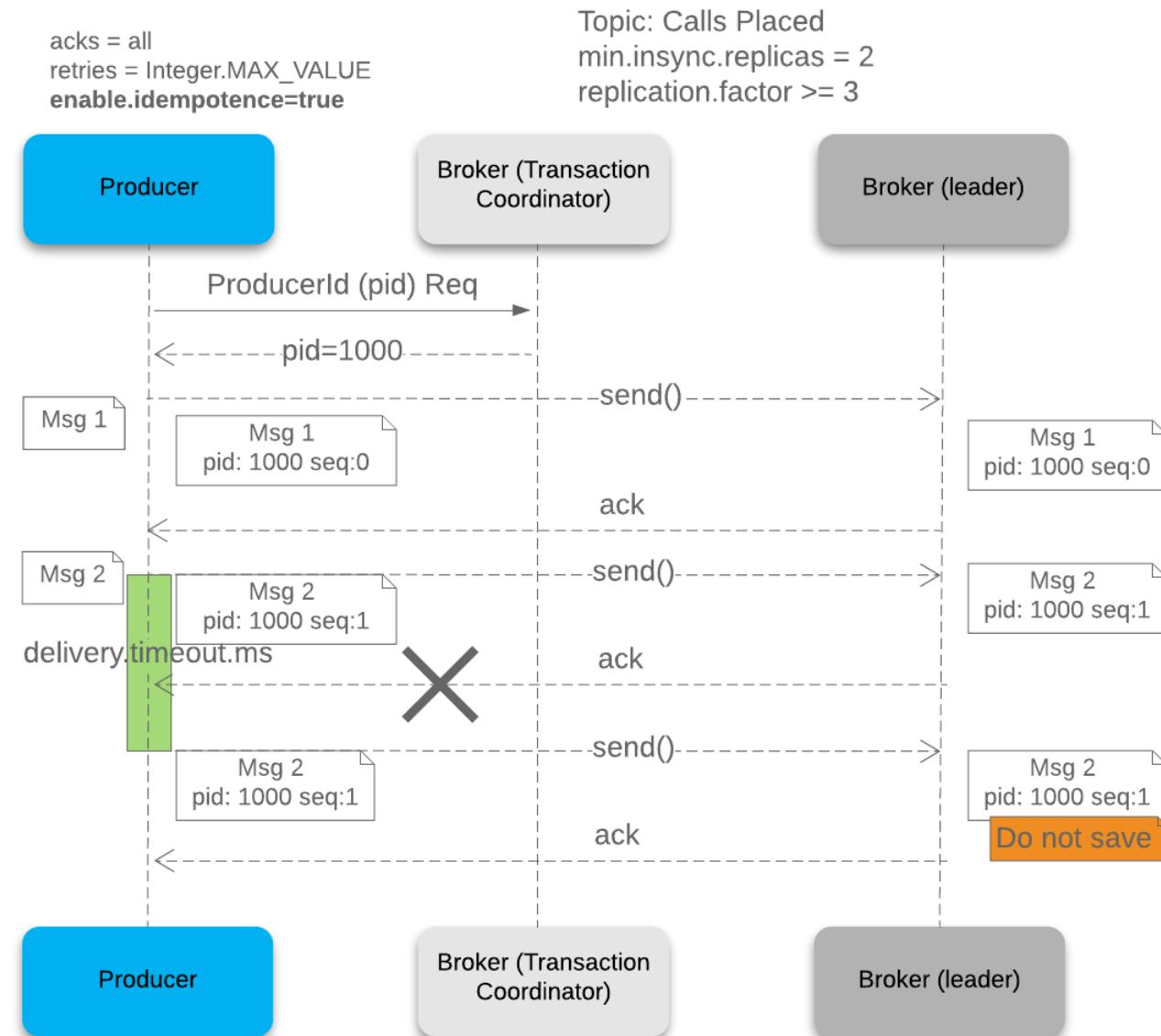
- Delivery Semantics
- Optional Instructor-Led Demonstration: ISRs vs. ACKs
- **Producer Delivery**
- Hands-On Exercise: Idempotent Producer
- Transactions
- Hands-On Exercise: Transactional Producers and Consumers
- Handling Consumer Failure
- Offset Management
- Hands-On Exercise: Detecting and Suppressing Duplicate Messages
- Hands-On Exercise: Handling Invalid Records
- Handling Producer Failure
- Essential Points

# Producer Delivery: Idempotent Producer (1)

---

- Helps prevent duplicate messages caused by producer's automatic re-send of messages
- Producers attach metadata to each message:
  - ProducerID : Unique ID automatically assigned to Producer
  - Epoch : Session ID
  - Sequence : Sequential number for each message in a topic+partition
- Brokers use metadata to identify messages which have already been saved
  - Drop the duplicate message
  - Return ack to producer
- Requires no changes in producer code
  - To enable, set `enable.idempotence` to true

# Producer Delivery: Idempotent Producer (2)



# Idempotent Producer: Recommended Settings

---

Option Name	Scope	Value
<code>enable.idempotence</code>	Producer	<code>true</code>
<code>retries</code>	Producer	<code>2147483647</code>
<code>acks</code>	Producer	<code>ALL</code>
<code>max.in.flight.requests.per.connection</code>	Producer	<code>1</code>

- **Idempotent producer incurs very little overhead**
- **Requires `acks=ALL`**
- **Requires `retries > 0`**

## Idempotent Producer: Caveats / Best Practice

---

- Idempotent producer prevents duplicate messages when producer *automatically* re-sends messages.
- Does not prevent *application logic* from sending duplicate messages
  1. Producer sends record #1 from RDBMS
  2. Producer crashes
  3. Another producer starts
  4. Producer resends record #1 from RDBMS
  5. Result: Duplicate record #1
- Only prevents duplicate messages sent from the same producer session
- Further advice at [Kafka Producer API Docs](#)

# Chapter Topics

---

## Improving Application Reliability

- Delivery Semantics
- Optional Instructor-Led Demonstration: ISRs vs. ACKs
- Producer Delivery
- **Hands-On Exercise: Idempotent Producer**
- Transactions
- Hands-On Exercise: Transactional Producers and Consumers
- 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: Idempotent Producer

---

- In this exercise, you will run idempotent producers and observe message metadata associated with messages sent by idempotent producers.
- Perform only the "Idempotent Producer" exercise.
- Exercise directory: `exercise-code/idempotence-transactions`

# Chapter Topics

---

## Improving Application Reliability

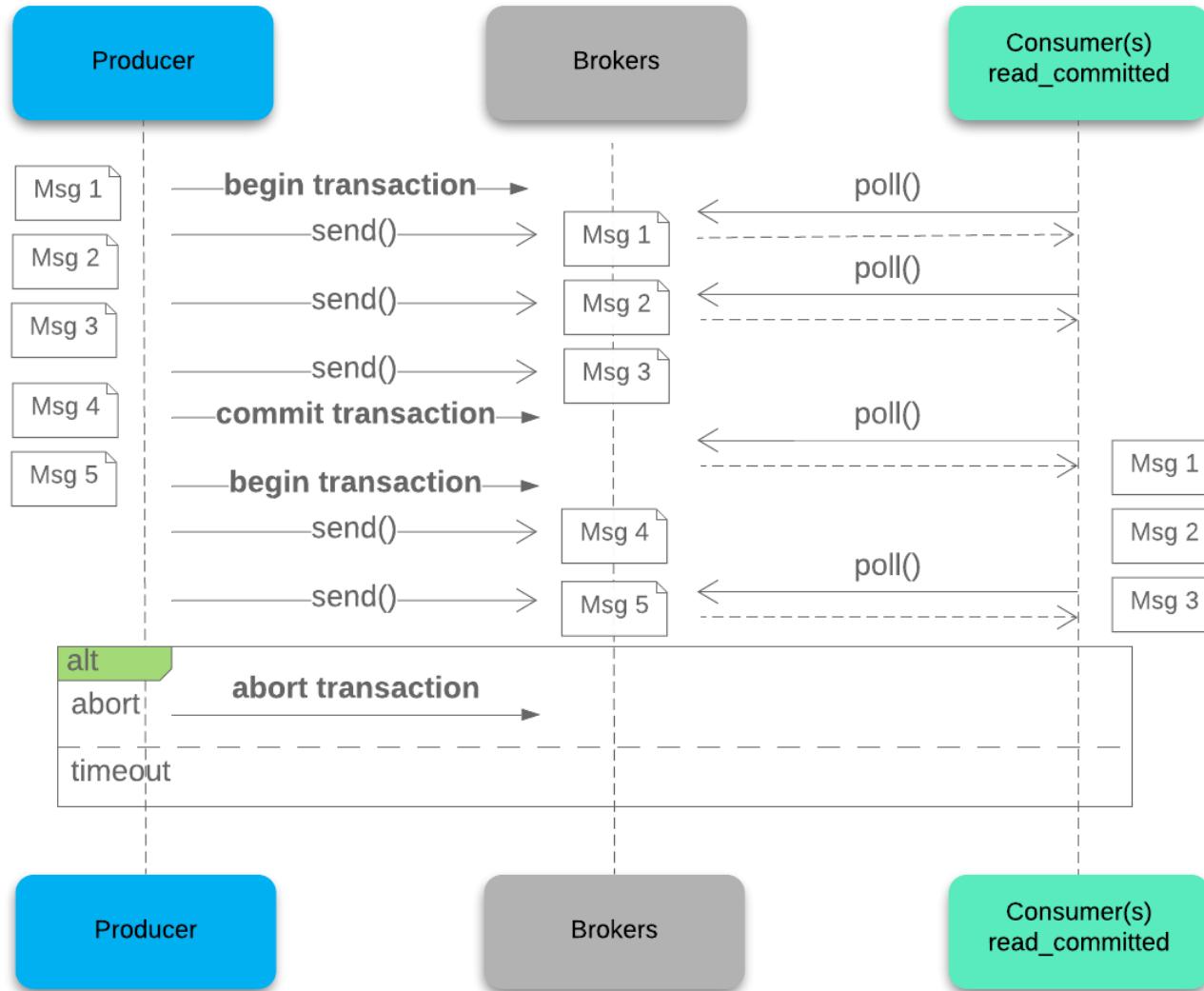
- Delivery Semantics
- Optional Instructor-Led Demonstration: ISRs vs. ACKs
- Producer Delivery
- Hands-On Exercise: Idempotent Producer
- **Transactions**
- Hands-On Exercise: Transactional Producers and Consumers
- Handling Consumer Failure
- Offset Management
- Hands-On Exercise: Detecting and Suppressing Duplicate Messages
- Hands-On Exercise: Handling Invalid Records
- Handling Producer Failure
- Essential Points

# Transactional Semantics / Exactly Once Semantics (EOS)

---

- **Provides atomic writes**
  - Producers send messages as part of a logical *transaction*
  - Transactions may span topics and partitions
  - Producers *commit* transactions at appropriate times
  - Committed messages are available to `read_committed` consumers
  - Duplicates are avoided by using a unique `transactional.id` for each instance of a producer
- **Consumers can choose to honor transaction status**
  - Consumers running in `read_committed` mode will see messages belonging to committed transactions and messages which do not belong to a transaction
  - Consumers running in `read_uncommitted` mode (default) will receive every message regardless of its transaction status

# Transactional Semantics - Overview

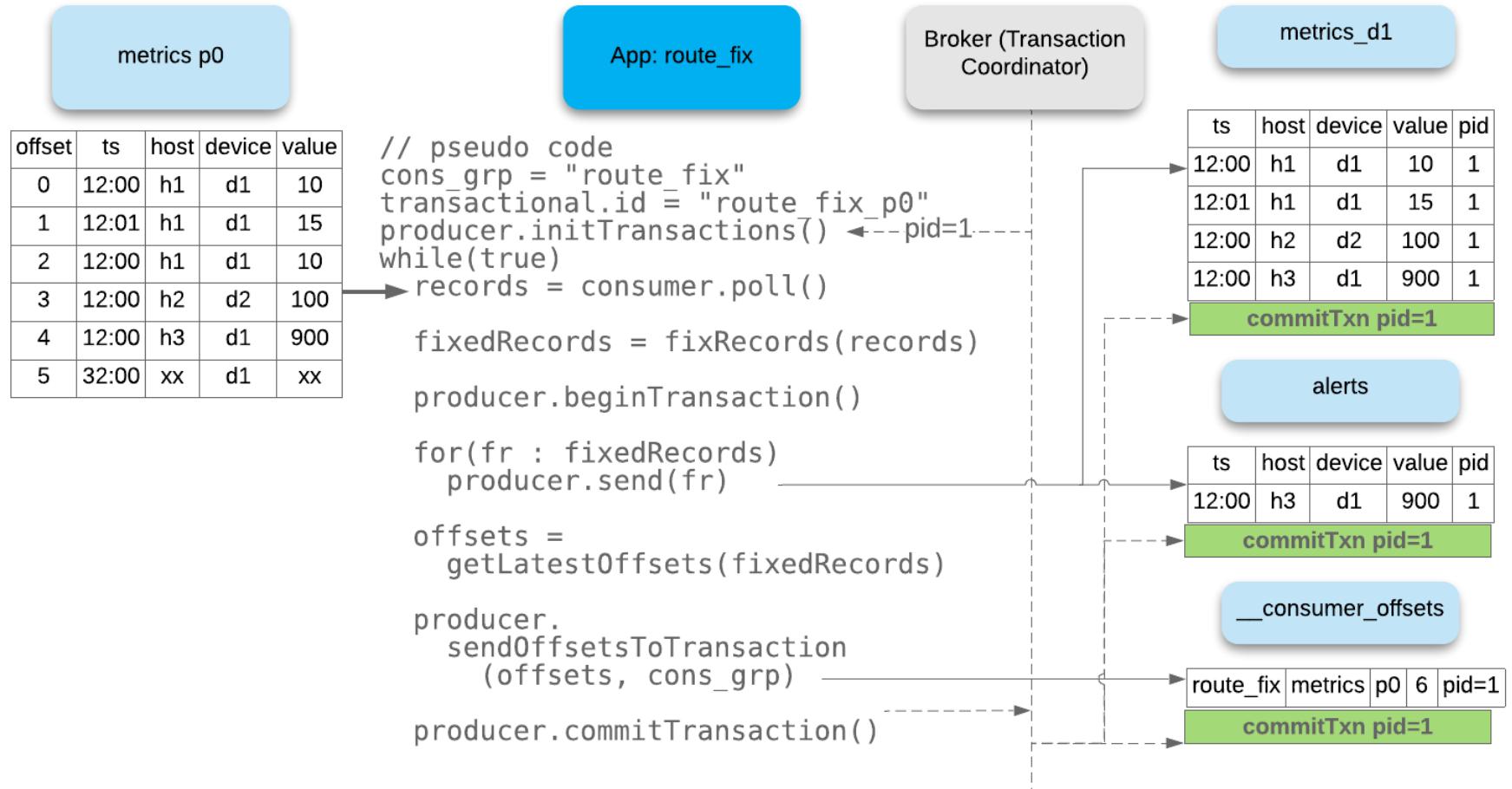


# Transactional Semantics - Use Cases

---

- **Read / Process / Write**
  - Applications which read messages from Kafka and produce messages to Kafka
  - Applications with strict requirements for processing messages once
- **Example: Metrics monitoring application**
  - Consumes from `metrics` topic
  - Routes messages to device-specific topics
  - Drops invalid messages
  - Creates alerts for out-of-bounds metrics (publishes new message to `alerts` topic)
  - Requirements:
    - Cannot produce duplicate alert messages
    - Cannot produce duplicate device-specific messages

# Read / Process / Write



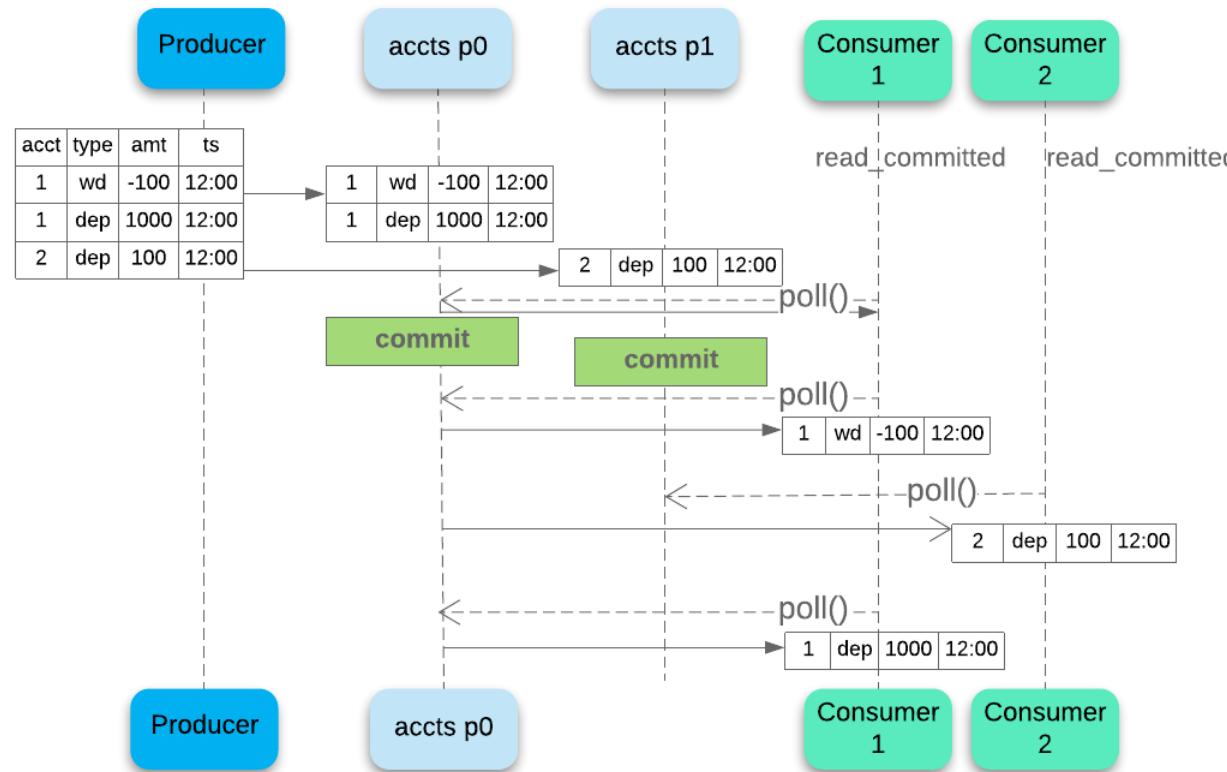
## Transactions Analogy: RDBMS vs. Kafka

---

```
SELECT * FROM accts WHERE id = 1 OR id = 2;
```

- **RDBMS: Queries are from a single "consumer" and span entire table(s)**
  - The above query will return all committed records
- **Kafka: "Queries" are (typically) issued from multiple consumers**
  - Each consumer is "querying" different partitions
  - Messages within the same transaction will probably be consumed by different consumers
  - Messages within the same transaction may be retrieved in different "queries" (poll loops)

# Transactions Analogy



## Producer Code

---

```
props.put("transactional.id", getMyTransactionalId());
producer.initTransactions();
try {
    producer.beginTransaction();
    sendSomeMessages();
    producer.commitTransaction();
} catch (ProducerFencedException |
OutOfOrderSequenceException | AuthorizationException e)
    producer.close();
} catch (KafkaException e) {
    // For all other exceptions, just abort the transaction
    and try again.
    producer.abortTransaction();
}
producer.close();
```

## Producer Code - Transactional ID

---

- Each instance of a producer must have a unique transactional.id
- Used to **fence** multiple instances of the same producer in case of crashes and timeouts

```
props.put("transactional.id", getMyTransactionId());
```

- Example
  - Producer initializes with transactional.id = X
  - Producer begins transaction
  - Sends msg 1, msg2
  - Producer crashes or stalls
  - Another instance of producer starts with transactional.id = X
  - Kafka aborts all open transactions from transactional.id = X
  - If previous instance of producer attempts to send messages, it receives a ProducerFencedException

# Recommendations for transactional.id

---

- Must be unique *per producer instance*
- General Ideas:
  - Use `client.id` + some "task id"
  - Designate a certain # of tasks which App will use
  - When an instance of a producer starts, request a taskId
- Worthwhile to explore various open-source projects for implementations:
  - Apache Flink: [Kafka Transactional EOS](#)
  - Kafka Streams: [Kafka Streams Exactly Once Design](#)

## Producer Code: Init Transactions

---

```
producer.initTransactions();
```

- **initTransactions** requests ProducerId from a *Transaction Coordinator*
  - Any broker may act as a transaction coordinator
  - Different brokers act as transaction coordinators for different producers
  - Transaction coordinators maintain transaction states, timeouts, ProducerIds
  - Transaction coordinator will *fence* a producer if another producer calls **initTransactions** using the same **transactional.id**

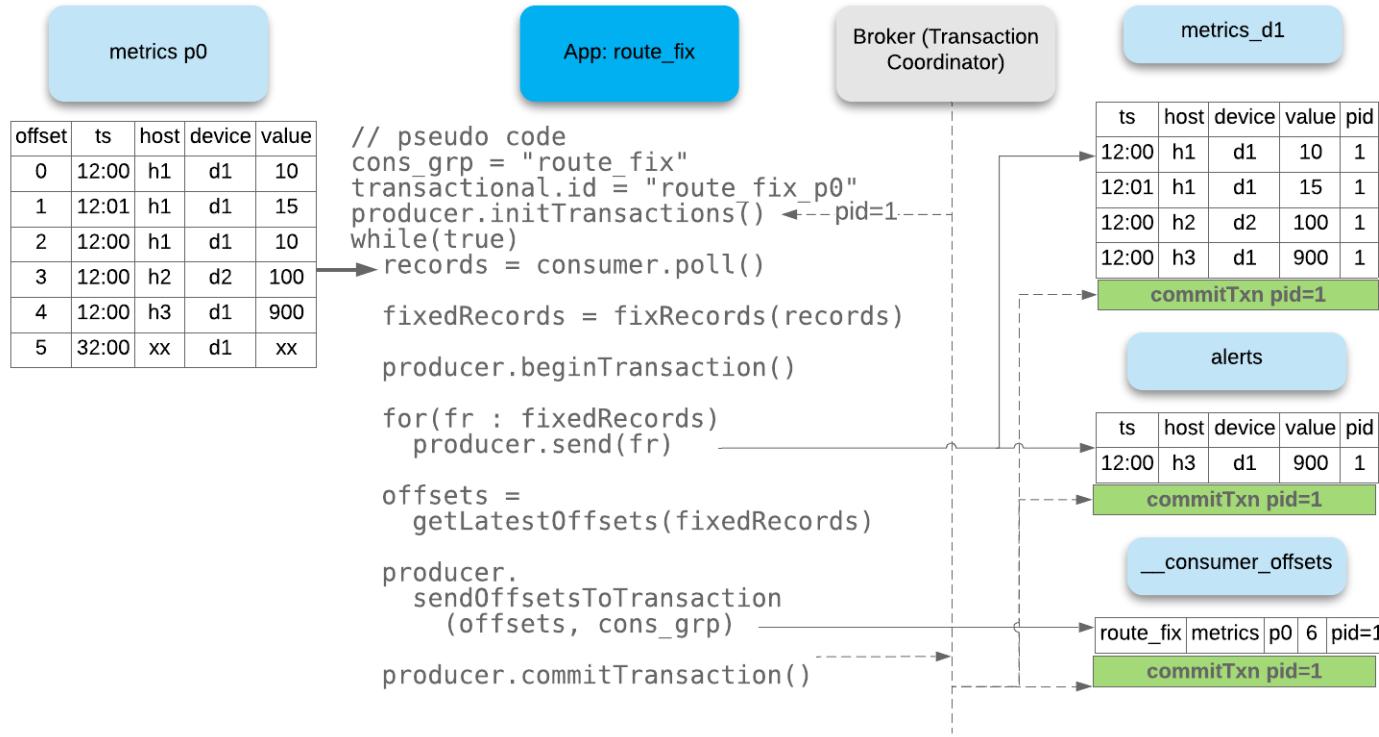
## Producer Code: Sending Transactional Messages

---

```
try {  
    producer.beginTransaction();  
    producer.send(...);  
    producer.send(...);  
    producer.send(...);  
    producer.commitTransaction();
```

- **Messages sent between `beginTransaction()` and `commitTransaction()` are part of the same transaction**
  - `commitTransaction()` is a synchronous operation
  - Waits for all messages sent to be acknowledged

# Recap:



- Use `producer.sendOffsetsToTransaction()` to commit offsets of messages consumed as part of current transaction
- Essentially "synchronizes" messages consumed and messages produced

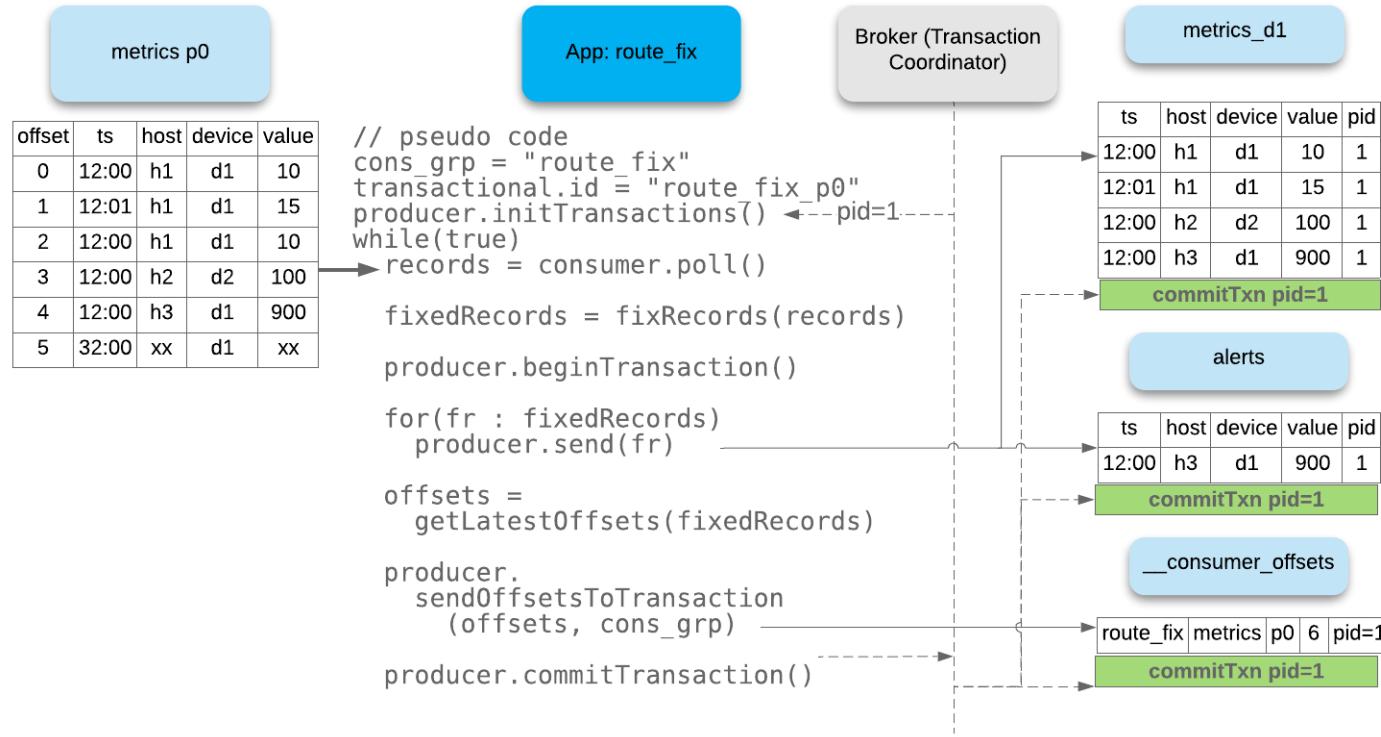
## Producer Code: Handling Errors

---

```
...
producer.commitTransaction();
} catch (ProducerFencedException | OutOfOrderSequenceException
| AuthorizationException e)
producer.close();
} catch (KafkaException e) {
// For all other exceptions, just abort the transaction and try
again.
producer.abortTransaction();
}
producer.close();
```

- If any `send()` calls in the transaction encounter irrecoverable errors, `commitTransaction()` will throw the last received exception immediately
- Not necessary to define a callback for `send()` or check the result of Future returned by `send()` when using transactions :-)

# Recap: Read / Process / Write



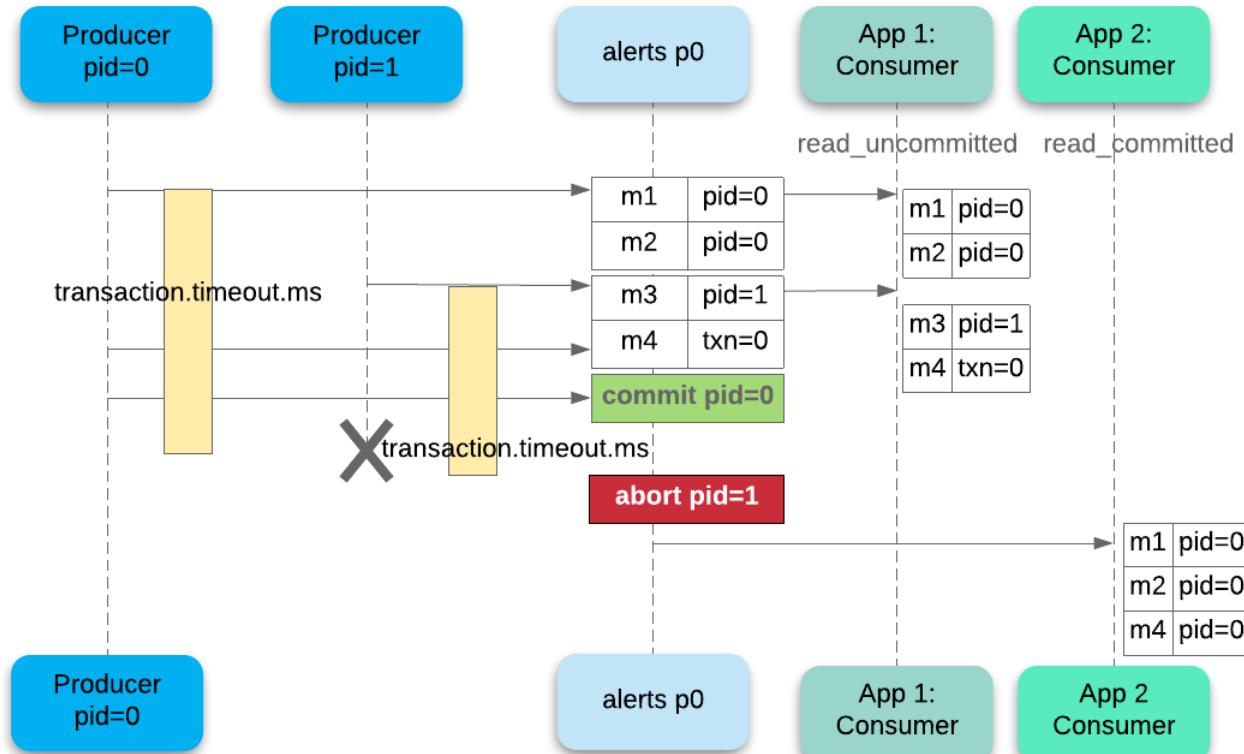
- Use `producer.sendOffsetsToTransaction()` to commit offsets of messages *consumed* as part of current transaction
- Essentially "synchronizes" messages consumed and messages produced

# Transactional Semantics: Consumer Code

```
Properties props = new Properties();
...
props.setProperty(ConsumerConfig.ISOLATION_LEVEL_CONFIG, "read_committed");
try (KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props)) {
    consumer.subscribe(topics);
    ConsumerRecords<String, String> records = consumer.poll(1000);
    ...
}
```

- Set `isolation.level` to `read_committed`
- Consumers will only be sent messages which are:
  - Non-transactional
  - Transactional and have been committed
- Default is `read_uncommitted`
  - `read_uncommitted` consumers will be sent *all* messages
  - Messages belonging to aborted transactions and uncommitted transactions will be sent regardless of status

# Transactional Semantics: Read Committed vs. Uncommitted



# Transactional Semantics: Best Practices

---

- Consider use-case: Are transactional semantics necessary?
- **read\_committed consumers will lag if a transactional producer lags**
  - Messages belonging to committed transactions will not be delivered to consumers if messages are "interleaved" with newer uncommitted messages in the same topic+partition!
- Consider sensible values for `transaction.timeout.ms`
  - Default is 60 seconds!
  - Too high will cause consumers to lag for `transaction.timeout.ms` if a producer fails or times out
  - Too low will cause transactions to be aborted prematurely
- Consider sensible number of messages which constitute a transaction
  - `commitTransaction()` will block producer code and hinder throughput
  - Balance logical transactions vs. overhead of `commitTransaction` and consumer lag

## Recommended Settings: Exactly once delivery

Option Name	Scope	Value
<code>transactional.id</code>	Producer	Unique value per producer <i>instance</i>
<code>transaction.timeout.ms</code>	Producer	See discussion on prev. slide
<code>enable.idempotence</code>	Producer	Automatically set to true when using transactions
<code>acks</code>	Producer	ALL (Required)
<code>retries</code>	Producer	(Must be non-zero)
<code>replication.factor</code>	Topic	3
<code>min.insync.replicas</code>	Topic	2
<code>isolation.level</code>	Consumer	<code>read_committed</code> to honor transaction status.

# Bibliography

---

The following offer more information on topics discussed in this section

- **Apache Kafka Exactly Once Delivery and Transactional Messaging in Kafka**
  - Implementation details, descriptive history for Exactly-Once Semantics (EoS)
  - Referenced by "separate document" link in KIP-98 (See Below)
- **KIP-98 Kafka Improvement Process (KIP) describing EoS**
- **Kafka Java API Docs:**
  - Kafka Producer Java API Docs (Examples of transactional producer code)
  - Kafka Consumer Java API Docs (See the "Reading Transactional Messages" section)
  - Isolation level / implementation for Consumers

# Chapter Topics

---

## Improving Application Reliability

- Delivery Semantics
- Optional Instructor-Led Demonstration: ISRs vs. ACKs
- Producer Delivery
- Hands-On Exercise: Idempotent Producer
- Transactions
- **Hands-On Exercise: Transactional Producers and Consumers**
- 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: Transactional Producers and Consumers

---

- Exercises include the following:
  - **Transactional Producers and Consumers**
    - Observing implementation and behavior of fast and slow transactional producers vs `read_committed` and `read_committed` consumers
  - **Crashy Producers**
    - Observing effects of producers which crash and stall consumption of messages by `read_committed` consumers
  - **Fenced Producers**
    - Observe effects of running two producers with the same `transactional.id` simultaneously
  - **Exercise directory: `exercise-code/idempotence-transactions`**

# Chapter Topics

---

## Improving Application Reliability

- Delivery Semantics
- Optional Instructor-Led Demonstration: ISRs vs. ACKs
- Producer Delivery
- Hands-On Exercise: Idempotent Producer
- Transactions
- Hands-On Exercise: Transactional Producers and Consumers
- **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

# Chapter Topics

---

## Improving Application Reliability

- Delivery Semantics
- Optional Instructor-Led Demonstration: ISRs vs. ACKs
- Producer Delivery
- Hands-On Exercise: Idempotent Producer
- Transactions
- Hands-On Exercise: Transactional Producers and Consumers
- Handling Consumer Failure
- **Offset Management**
- Hands-On Exercise: Detecting and Suppressing Duplicate Messages
- Hands-On Exercise: Handling Invalid Records
- Handling Producer Failure
- Essential Points

# Consumer Groups: Offset Management

- Most Kafka clients use Kafka's built-in Offset Management

Partition	Offsets	Log-End Offset
0	0,1,2	2
1	0,1,2,3	3
2	0,1,2,3,4	4

Topic: \_\_consumer\_offsets

Group	Topic-Partition	Last Committed Offset
Dept1	Calls-0	1
Dept1	Calls-1	1
Dept1	Calls-2	2

group.id=Dept1  
client.id=Dept1Foo

Cons 1 consumer-id:  
Dept1Foo-a8fb2ac..  
host: /10.0.9.11  
topic-part: Calls-0  
lag: 1

Cons 2 consumer-id:  
Dept1Foo-302bcdd..  
host: /10.0.9.11  
topic-part: Calls-1  
lag: 2

Cons 3 consumer-id:  
Dept1Foo-ff2ab93..  
host: /10.0.9.12  
topic-part: Calls-2  
lag: 2

# 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()`

# Chapter Topics

---

## Improving Application Reliability

- Delivery Semantics
- Optional Instructor-Led Demonstration: ISRs vs. ACKs
- Producer Delivery
- Hands-On Exercise: Idempotent Producer
- Transactions
- Hands-On Exercise: Transactional Producers and Consumers
- 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

- Delivery Semantics
- Optional Instructor-Led Demonstration: ISRs vs. ACKs
- Producer Delivery
- Hands-On Exercise: Idempotent Producer
- Transactions
- Hands-On Exercise: Transactional Producers and Consumers
- 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

- Delivery Semantics
- Optional Instructor-Led Demonstration: ISRs vs. ACKs
- Producer Delivery
- Hands-On Exercise: Idempotent Producer
- Transactions
- Hands-On Exercise: Transactional Producers and Consumers
- 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

- Delivery Semantics
- Optional Instructor-Led Demonstration: ISRs vs. ACKs
- Producer Delivery
- Hands-On Exercise: Idempotent Producer
- Transactions
- Hands-On Exercise: Transactional Producers and Consumers
- 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

- Apache Kafka Documentation

- See the Configuration section for Kafka Producer/Broker/Consumer configuration settings
  - See the APIs section for Examples of Apache Kafka Producer and Consumer API



# Analyzing Kafka Clusters with SMM

---

Chapter 10

# Course Chapters

---

- Introduction
- Kafka Overview
- Deploying Apache Kafka
- Kafka Command Line Basics
- Using Streams Messaging Manager (SMM)
- Kafka Java API Basics
- Improving Availability through Replication
- Improving Application Scalability
- Improving Application Reliability
- **Analyzing Kafka Clusters with SMM**
- Monitoring Kafka
- Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka Cluster
- Conclusion

# Analyze Kafka Clusters

---

By the end of this chapter, you will be able to

- **Monitor end-to-end latency**
- **Create and manage notifications**
- **Configure and manage alert policies**
- **Identify common use cases**

# Chapter Topics

---

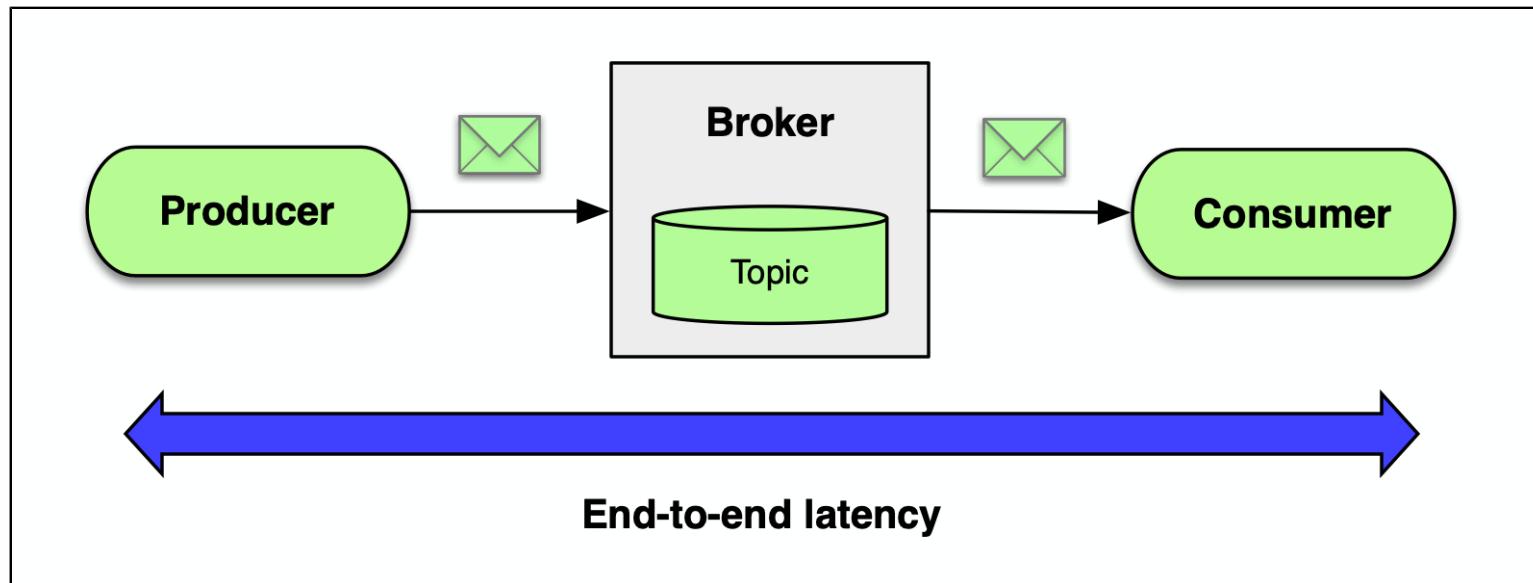
## Analyzing Kafka Clusters with SMM

- **End-to-End Latency**
- Notifiers
- Alert Policies
- Use Cases
- Essential Points

## End-To-End Latency

---

- End-to-end latency is the time it takes for a record produced to a Kafka topic or broker to be fetched by a consumer



# Questions: End-To-End Latency

---

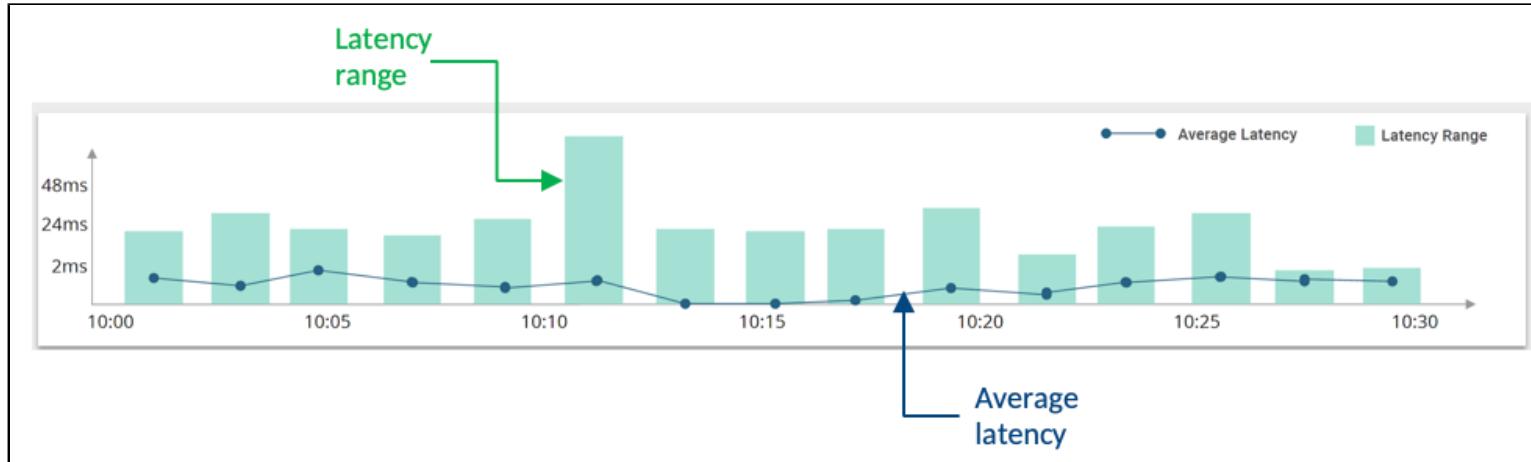
## Questions

*For a given topic, over a time window, find the average latency for messages in that time window*

- For the last five minutes, what is the average latency of messages consumed after being produced?
- For the last hour, which topics are not being consumed fast enough?

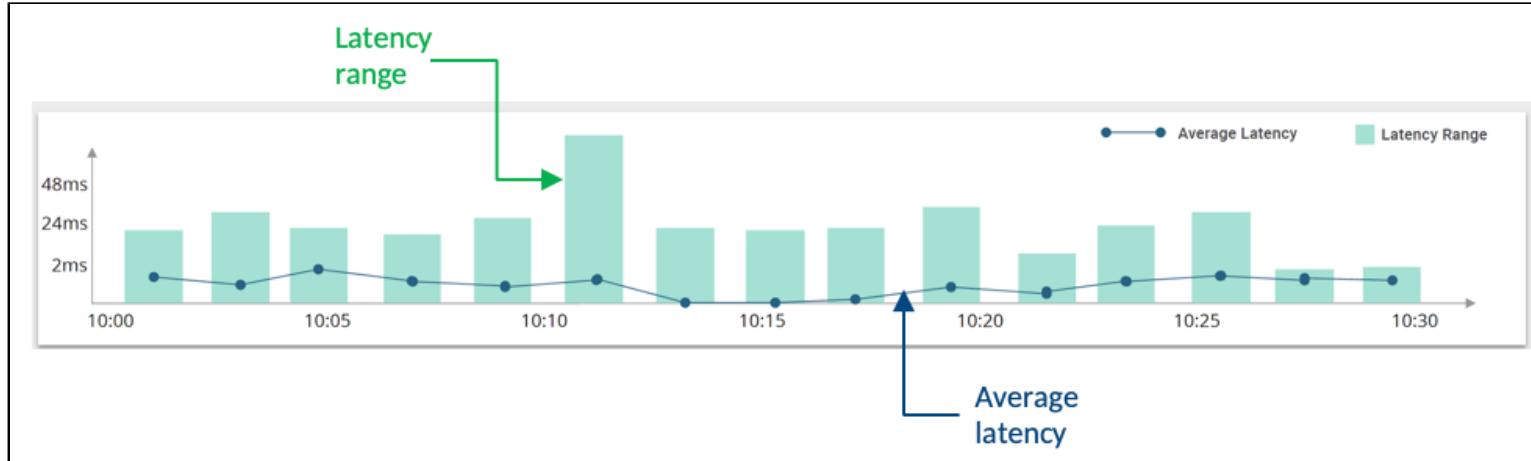
- Analyzing end-to-end latency allows you to identify delays that a message can encounter while moving through the system
- Depending on the root cause, solutions can include
  - Modifying Kafka client and broker configuration parameters
  - Adding or removing partitions or brokers

# End-To-End Latency Graph



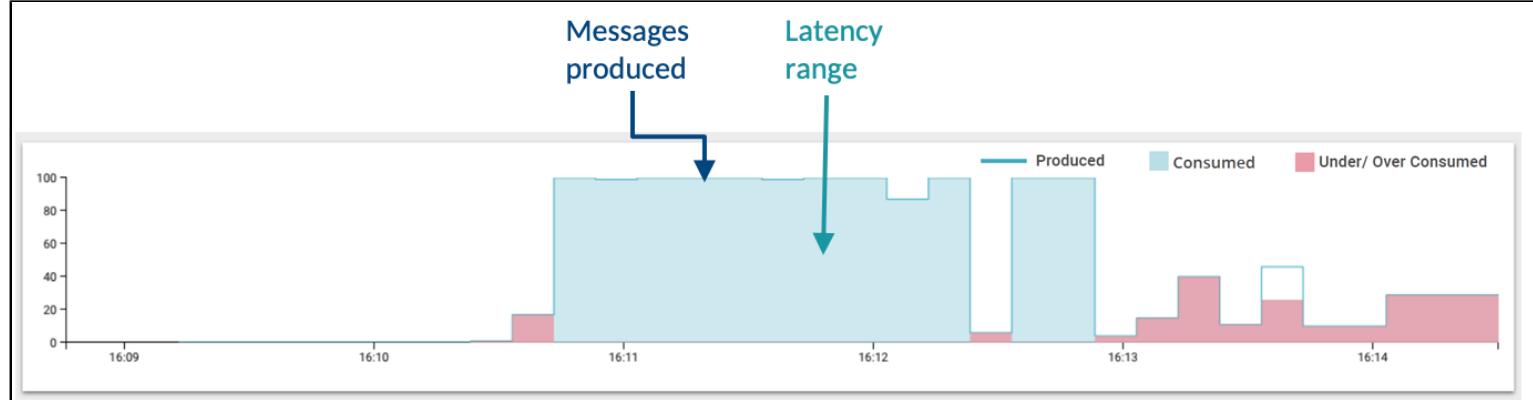
- Displays the latency range and average latency for a selected topic over a given time range
  - Green vertical bars: latency range
  - Dotted line: average latency

# End-To-End Latency Metrics



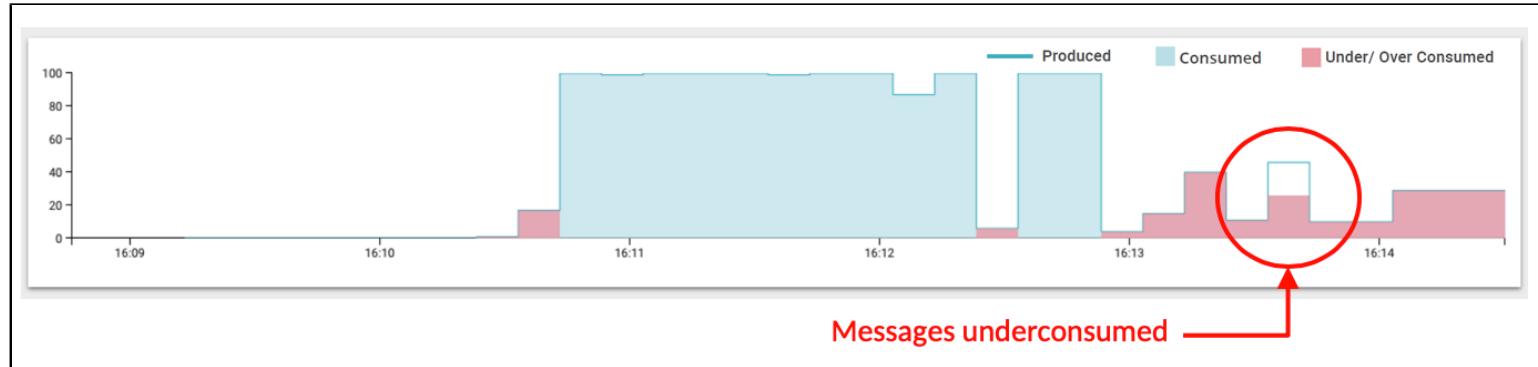
- Two different granularities
  - Data in last 24 hours shown in 30 second intervals
  - Older data shown in 15 minute intervals
- By default, 30 second metrics are stored for 24 hours and 15 minute metrics are stored for two weeks

# Messages Consumed Graph



- Displays message counts for messages produced and messages consumed for a selected topic over a given time range

# Under-Consumed and Over-Consumed Message



- Discrepancies between the produced and consumed message count is highlighted in red
  - Underconsumption of messages: if the red area is less than the messages produced
  - Overconsumption of messages: If the red area is greater than the messages produced

# End-To-End Latency Configuration

---

- Enable interceptors to display end-to-end metrics for producers and consumers
- General steps:
  1. Add the `monitoring-interceptors` jar to the classpath of the application
  2. For a consumer, add the `interceptor.classes` property to configuration that gets passed to the `KafkaConsumer` constructor
  3. For a producer, add the `interceptor.classes` property to configuration that gets passed to the `KafkaProducer` constructor
- [Cloudera SMM Documentation: Enabling Interceptors](#) for complete steps for enabling interceptors
- Note that for all other displayed metrics, SMM queries the REST API and no further configuration is needed

# Chapter Topics

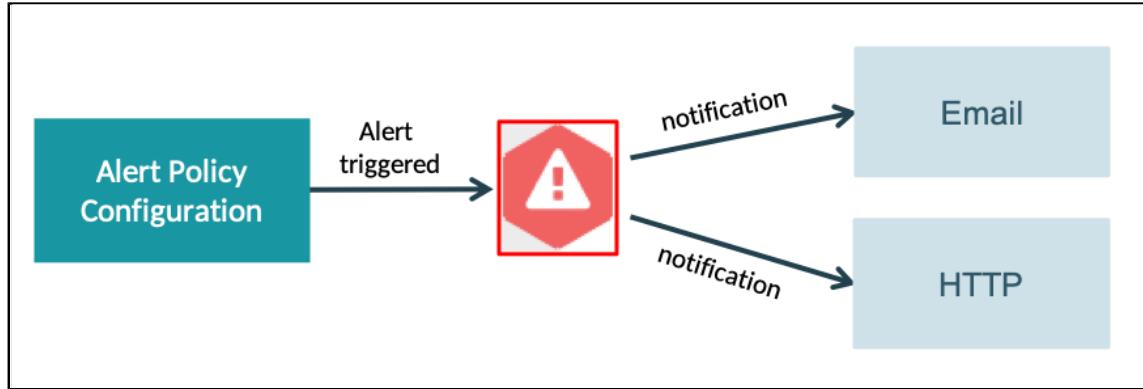
---

## Analyzing Kafka Clusters with SMM

- End-to-End Latency
- **Notifiers**
- Alert Policies
- Use Cases
- Essential Points

# Overview: Alert Policies and Notifiers

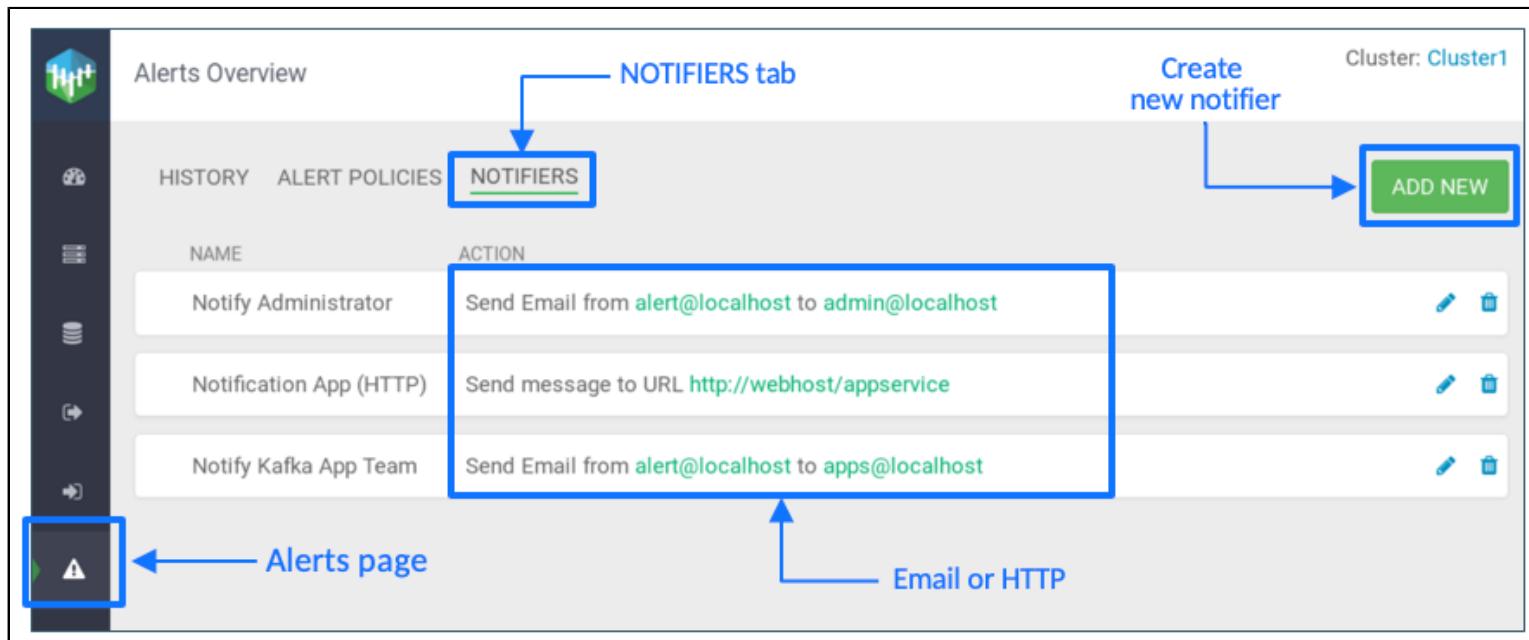
---



- **An alert policy specifies**
  - A set of conditions based on Kafka entities
  - One or more notifiers
- **A notifier**
  - Communicates alert information to appropriate recipients
  - Is automatically sent when an alert is triggered
  - Is one of two types: **Email** or **HTTP**

# Create and Manage Notifiers

- Managed on the **Alerts page**
- **NOTIFIERS tab** lists all notifiers
- Click **Add New** to create a new notifier



## Notifier Common Settings

---

- **NAME** and **DESCRIPTION** are required
- **NAME** must be unique
- Select **Http** or **Email** for **PROVIDER**

Notifier

NAME  
Notify DevOps

DESCRIPTION  
DevOps group notification

PROVIDER  
Email

Http

Email

# Notifier Rate Limit

---

NOTIFIER RATE LIMIT	
COUNT	DURATION
<input type="text" value="1"/>	<input type="text" value="HOUR"/>

- **NOTIFIER RATE LIMIT COUNT**
  - Number of notifications sent until alert is cleared
- **NOTIFIER RATE LIMIT DURATION**
  - Number of notifications for a given duration
  - Specify **SECOND**, **MINUTE**, or **HOUR**

# HTTP Configuration

---

- **URL:** Target service URL
- **CONNECTION TIMEOUT:** Connection timeout in milliseconds for creating the initial connection
- **READ TIMEOUT:** Read timeout in milliseconds for waiting to read data

Notifier

NAME  
Notification App (HTTP)

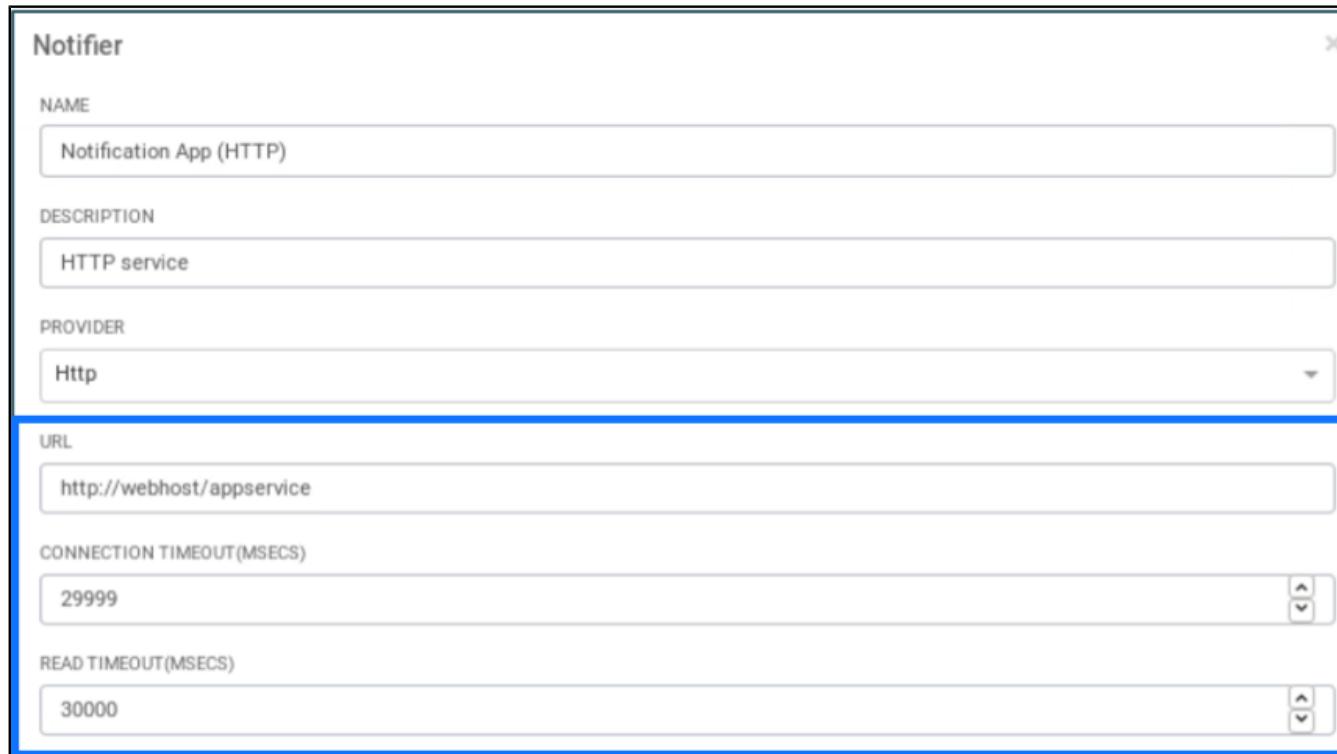
DESCRIPTION  
HTTP service

PROVIDER  
Http

URL  
http://webhost/appservice

CONNECTION TIMEOUT(MSECS)  
29999

READ TIMEOUT(MSECS)  
30000



# Email Configuration SMTP Server

- TO ADDRESS can be one or more email addresses
- Mail server parameters include
  - USERNAME: Username for SMTP
  - PASSWORD: Password for SMTP
  - SMTP HOSTNAME: SMTP server
  - SMTP PORT: SMTP port

FROM ADDRESS  
admin@localhost

TO ADDRESS  
alert@localhost

USERNAME  
smm-alert

PASSWORD  
\*\*\*\*\*

SMTP HOSTNAME  
localhost

SMTP PORT  
25

Mail server parameters

# Chapter Topics

---

## Analyzing Kafka Clusters with SMM

- End-to-End Latency
- Notifiers
- **Alert Policies**
- Use Cases
- Essential Points

# Alert Policies

---

- **An alert policy**
  - Triggers an alert based on the conditions configured
  - Specifies notifications using one or more notifiers
- **Use alerts to**
  - Monitor the health of different Kafka entity types, latency, and Kafka cluster replications
  - Identify and troubleshoot problems

## Alert Policies: History

- The HISTORY tab displays a list of alerts that have been RAISED or CLEARED
- ACTIONS > Mark All As Read to mark all the alerts as read
- Dismiss icon for each alert to mark the alert as read

HISTORY	ALERT POLICIES	NOTIFIERS					ACTIONS ▾
Title	Timestamp ▾	Component name	Type	State	Payload		
⚠ Alert: Producer Time Laps...	44s ago	ANY	PRODUCER	RAISED	Alert policy : "ALERT IF ( ANY PRODUCER MILLISECONDS_LAPSED_SINCE_PROD... <a href="#">show more</a>	0	
⚠ Alert: Java Consumer Acti...	53s ago	41	BROKER	RAISED	Alert policy : "ALERT IF ( BROKER (name='41') BYTES_OUT_PER_SEC < 1000... <a href="#">show more</a>	0	
⚠ Alert: Producer Time Laps...	5m 44s ago	ANY	PRODUCER	RAISED	Alert policy : "ALERT IF ( ANY PRODUCER MILLISECONDS_LAPSED_SINCE_PROD... <a href="#">show more</a>	0	
⚠ Alert: Java Consumer Acti...	5m 53s ago	41	BROKER	RAISED	Alert policy : "ALERT IF ( BROKER (name='41') BYTES_OUT_PER_SEC < 1000... <a href="#">show more</a>	0	
⚠ Alert: Producer Time Laps...	10m 44s ago	ANY	PRODUCER	RAISED	Alert policy : "ALERT IF ( ANY PRODUCER MILLISECONDS_LAPSED_SINCE_PROD... <a href="#">show more</a>	0	
⚠ Alert: Java Consumer Acti...	10m 53s ago	41	BROKER	RAISED	Alert policy : "ALERT IF ( BROKER (name='41') BYTES_OUT_PER_SEC < 1000... <a href="#">show more</a>	0	

# Create an Alert Policy

- The **ALERT POLICIES** tab displays a list of all alert policy definitions
- Click **Add New** to create a new alert

The screenshot shows the 'ALERT POLICIES' tab in a monitoring application. At the top, there are three tabs: 'HISTORY', 'ALERT POLICIES' (which is selected and highlighted in green), and 'NOTIFIERS'. Below the tabs, there are four columns: 'NAME', 'CONDITION', 'DESCRIPTION', and 'ENABLE'. Three alert policies are listed:

NAME	CONDITION	DESCRIPTION	ENABLE
Producer Time Lapsed	IF ANY Producer has MILLISECONDS LAPSED SINCE PRODUCER WAS ACTIVE > 5000	Alert when any producer t...	<input checked="" type="checkbox"/>
Java Consumer Active	IF Broker: 41 has BYTES OUT PER SEC < 1000 AND Topic: call_failed has BYTES OUT PER SEC < 1000	Alert policy to monitor co...	<input checked="" type="checkbox"/>
Consumer Group Lag	IF ANY Consumer has CONSUMER GROUP LAG = 4	Alert when lag > 4	<input checked="" type="checkbox"/>

A blue arrow points from the text 'Create new alert policy' to the green 'ADD NEW' button at the bottom right of the table.

# Alert Policy Configuration

- Alert policy sections
  - General: Name, description, and execution settings
  - Policy: Alert condition
  - Action: One or more notifiers
  - Preview: Displays the alert policy configuration

The screenshot shows the 'Alert Policy' configuration dialog box. It is divided into four main sections:

- General**: Contains fields for NAME ('Consumer Group Lag'), DESCRIPTION ('Alert when lag > 4'), EXECUTION INTERVAL IN SECONDS (set to 60), EXECUTION DELAY IN SECONDS (set to 300), and an ENABLE toggle switch.
- Condition**: Contains an 'IF...' dropdown set to 'Consumer' and a 'TARGET NAME' dropdown set to 'ANY'. Below these are 'ATTRIBUTE', 'CONDITION', and 'VALUE' fields, with 'CONSUMER GROUP LAG' selected, '=' chosen as the condition, and '4' entered as the value.
- Notifiers**: Contains a 'NOTIFICATION' dropdown with two items: 'Notify Kafka App Team' and 'Notification App (HTTP)'.
- Preview**: Displays the resulting alert configuration: 'IF ANY CONSUMER has CONSUMER GROUP LAG = 4 THEN notify by Notify Kafka App Team, Notification App (HTTP)'. At the bottom right are 'Cancel' and 'Save' buttons.

# Alert Policy Execution

- **EXECUTION INTERVAL IN SECONDS:** Time for executing the alert policy periodically after the given time interval
- **EXECUTION DELAY IN SECONDS:** Time of delay for execution of the alert policy when the alert policy triggered an alert

The screenshot shows a configuration dialog titled "Alert Policy". It includes fields for "NAME" (set to "Consumer Group Lag") and "DESCRIPTION" (set to "Alert when lag > 4"). At the bottom, there are two input fields: "EXECUTION INTERVAL IN SECONDS" (set to "60") and "EXECUTION DELAY IN SECONDS" (set to "300"). To the right of these fields is an "ENABLE" toggle switch, which is turned on (blue). A blue rectangular box highlights the "EXECUTION INTERVAL IN SECONDS" and "EXECUTION DELAY IN SECONDS" fields.

## Alert Condition: What Can You Monitor?

---

- Component types used in the alert policy include
  - Broker
  - Consumer
  - Producer
  - Topic
  - Latency
  - Cluster
  - Cluster replication
- Each component type has associated attributes that can be monitored
- Can specify multiple conditions that combine component types

## Alert Condition: Simple

- IF *component type name* has *attribute condition* THEN notify by *notification(s)*
  - Where component type name can be a specified component name, ALL components, or ANY component

Policy

COMPONENT TYPE	TARGET NAME
IF... Consumer	ANY
ATTRIBUTE	CONDITION VALUE
CONSUMER GROUP LAG	= 4

Action

NOTIFICATION

x Notify Kafka App Team x Notification App (HTTP)

Preview displays alert payload

Preview

IF ANY CONSUMER has CONSUMER GROUP LAG = 4 THEN notify by Notify Kafka App Team, Notification App (HTTP)

## Alert Condition: Complex

- Click the green + icon to add condition to the current component type
- Click the blue + icon to add another component type to the condition

Policy

COMPONENT TYPE	TARGET NAME
IF... Broker	41

ATTRIBUTE	CONDITION	VALUE
BYTES OUT PER SEC 	< =	1000 true

Add/remove condition

WITH	Topic	call_failed
ATTRIBUTE	BYTES OUT PER SEC	<

Add/remove condition

# Alert Payload

- Specifies the alert policy and condition

The screenshot shows a user interface for managing alerts. At the top, there are tabs for HISTORY, ALERT POLICIES, and NOTIFIERS, with HISTORY selected. On the right, there is a green ACTIONS button. The main area displays a table of alerts:

Title	Timestamp	Component name	Type	State	Payload
⚠ Alert: Java Consumer Acti...	4s ago	41	BROKER	RAISED	Alert policy : "ALERT IF ( BROKER (name="41") BYTES_OUT_PER_SEC < 1000... show m
⚠ Alert: Producer Time Laps...	4m 44s ago	ANY	PRODUCER	RAISED	Alert policy : "ALERT IF ( ANY PRODUCER MILLISECONDS_LAPSED_SINCE_PROD... sh
⚠ Alert: Java Consumer Acti...					ame="41") BYTES_OUT_PER_SEC < 1000... show m
⚠ Alert: Producer Time Laps...					UCER MILLISECONDS_LAPSED_SINCE_PROD... sh
⚠ Alert: Java Consumer Acti...					ame="41") BYTES_OUT_PER_SEC < 1000... show m

A blue arrow points from the "Payload" column of the first alert row down to a detailed payload view. This view contains the following information:

**Payload**

```
Alert policy : "ALERT IF ( BROKER (name="41") BYTES_OUT_PER_SEC < 1000 || IS_BROKER_DOWN = true ) WITH ( TOPIC (name="call_failed") BYTES_OUT_PER_SEC < 1000 )" has been evaluated to true
```

Condition : "BYTES\_OUT\_PER\_SEC<1000||IS\_BROKER\_DOWN=true" has been evaluated to true for following BROKERS

- BROKER = "41" had following attribute values
  - \* BYTES\_OUT\_PER\_SEC = 116.00823
  - \* IS\_BROKER\_DOWN = false

Condition : "BYTES\_OUT\_PER\_SEC<1000" has been evaluated to true for following TOPICS

- TOPIC = "call\_failed" had following attribute values
  - \* BYTES\_OUT\_PER\_SEC = 0.0

Ok

# Alert Display

- Alerts that are triggered are displayed in SMM with the relevant Kafka entity
- Displays same alert payload information as on the **ALERT HISTORY** tab on the **Alerts page**

The screenshot shows the Cloudera Manager Metrics interface for the topic 'call\_failed'. The top navigation bar includes 'Topics / call\_failed', 'METRICS' (which is selected), 'DATA EXPLORER', 'CONFIGS', and 'LATENCY'. Below this, the 'Producers (1)' section displays a single producer named '41' with a replication factor of 1, 1 inSync replica, and 1,000 total messages. A message card for 'client1' shows 1k messages. An alert message is present: 'Alert: Java Consumer Active' with the policy 'call\_failed Alert policy : "ALERT IF ( BROKER (name="41") BYTES\_OUT\_PER\_SEC < 1000 ) WITH ( TOPIC (name="call\_failed") BYTES\_OUT\_PER\_SEC < 1000 )"' evaluated to true for broker 41 at 2h 23m 26s ago. A blue arrow points from this card to a larger, detailed view of the alert payload on the right.

**Alert: Java Consumer Active**

**call\_failed** Alert policy : "ALERT IF ( BROKER (name="41") BYTES\_OUT\_PER\_SEC < 1000 ) WITH ( TOPIC (name="call\_failed") BYTES\_OUT\_PER\_SEC < 1000 )" has been evaluated to true Condition : "BYTES\_OUT\_PER\_SEC<1000" has been evaluated to true for following BROKERS - BROKER = "41" had following attribute values \* BYTES\_OUT\_PER\_SEC = 725.7362 Condition : "BYTES\_OUT\_PER\_SEC<1000" has been evaluated to true for following TOPICS - TOPIC = "call\_failed" had following attribute values \* BYTES\_OUT\_PER\_SEC = 0.0

2h 4m 19s ago [Hide](#)

# Chapter Topics

---

## Analyzing Kafka Clusters with SMM

- End-to-End Latency
- Notifiers
- Alert Policies
- **Use Cases**
- Essential Points

## Scenario: Topic BYTES OUT PER SECOND

---

- **Scenario: BYTES OUT PER SECOND is low compared to BYTES IN PER SECOND**
- **Problem: Bytes out too low might indicate that the consumer is lagging or inactive**
- **Suggestions:**
  - Create an alert policy to trigger if it goes below an expected value
  - Create an alert policy to trigger if the consumer becomes inactive

## Scenario: Consumer Group LAG

---

- **Scenario: Consumer group LAG is high**
- **Problem: A high value might indicate that the consumer group cannot keep up with the amount of messages or it is inactive**
- **Suggestions:**
  - Create an alert policy to trigger if LAG goes above an expected value
  - Create an alert polity to trigger if the consumer becomes inactive
  - Add another consumer to the consumer group

## Scenario: Producer is Not Active

---

- **Scenario: Producer is not active**
- **Problem: A low or zero value for the Topic's BYTES IN PER SECOND can indicate that the producer is not active**
- **Suggestion:**
  - Create an alert policy to trigger if a producer becomes inactive

## Scenario: OUT OF SYNC REPLICA COUNT

---

- **Scenario:** The topic's replica count is out of sync
- **Problem:**
  - The producer's throughput to the leader replica is too high
  - The follower `replica.fetch.wait.max.ms` was configured larger than `replica.lag.time.max.ms`
- **Suggestions:**
  - To reduce the overall number of out of sync replica count, try reducing the producer's batch size to avoid sending too many messages per batch
  - Check if `replica.fetch.wait.max.ms` was configured larger than `replica.lag.time.max.ms`

# Chapter Topics

---

## Analyzing Kafka Clusters with SMM

- End-to-End Latency
- Notifiers
- Alert Policies
- Use Cases
- Essential Points

## Essential Points

---

- End-to-end latency identifies lagging consumers and any discrepancies in the produced and consumed message count
- An alert policy triggers an alert based on conditions configured
- Alert notifications are sent via email or HTTP
- Alert payloads are displayed in the alert history and as a notification on the respective SMM page for the Kafka entity

# Bibliography

---

The following offer more information on topics discussed in this chapter

- [Cloudera SMM Documentation: End to End Latency Overview](#)
- [Cloudera SMM Documentation: Managing Alert Policies and Notifiers](#)



# Monitoring Kafka

---

Chapter 11

# Course Chapters

---

- Introduction
- Kafka Overview
- Deploying Apache Kafka
- Kafka Command Line Basics
- Using Streams Messaging Manager (SMM)
- Kafka Java API Basics
- Improving Availability through Replication
- Improving Application Scalability
- Improving Application Reliability
- Analyzing Kafka Clusters with SMM
- **Monitoring Kafka**
- Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka Cluster
- Conclusion

# Monitoring Kafka Overview

---

**By the end of this chapter, you will be able to**

- **Identify key events, metrics, and resources**
- **Discuss available monitoring tools in Cloudera Manager**
- **View charts in Cloudera Manager**
- **Investigate issues by viewing Kafka service logs**
- **Diagnose service failures**

# Chapter Topics

---

## Monitoring Kafka

- **Monitoring Overview**
- Monitoring using Cloudera Manager
- Charts and Reports in CM
- Monitoring Recommendations
- Metrics for Troubleshooting
- Diagnosing Service Failure
- Hands-On Exercise: Monitoring Kafka
- Essential Points

# Monitoring Overview

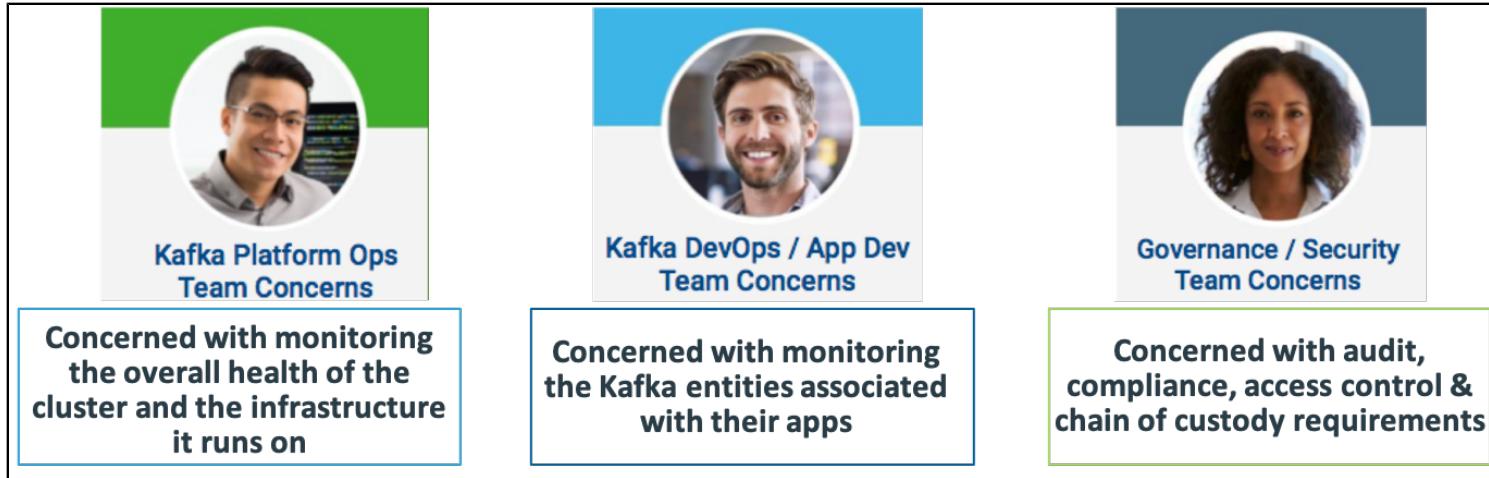
---

- **Monitor the health and performance of your cluster**
  - Monitor cluster health
  - Identify configuration issues
  - Track metrics and resource usage with charts and dashboards
  - View event logs
  - Generate alerts
  - Audit Cloudera Manager events
  - Generate reports

# Three Personas

---

- Both CM and SMM address the distinct needs of these three personas



# Platform Operations Team



Do I have any offline topic partitions?	How many total active producers/consumers is there currently?	Which producers are generating the most data right now?
Which consumer group us falling behind the most?	Which producers are generating the most data right now?	Are any of my brokers running out of disk space?
Are any of my brokers down?	Are there any skewed partitions for a broker?	How of the cluster is being used, how much capacity do I have available per broker / cluster?
Which producers are generating the most data right now?	How many total active producers and consumers exist now?	Which partitions are located on each broker?
What is the throughput in/out for a given partition on that broker?	Are any of my brokers running hot? Which broker has the highest throughput in and and rates?	Which of my topics has produced/consumers the most messages over the last N minutes/hours?
What hosts are my brokers located on?	How many total topics does my Kafka cluster have?	
Are all my replicas in my topic in-synch?		

# Application Development Team



Kafka DevOps / App Dev  
Team Concerns

Find all entities (producer, consumers, topics) associated with my app.

What brokers holds the partitions for my app topics?

What is the total number of messages into my topic over the last N minutes/hours?

Are there consumers in a consumer-group for a given topic slow/falling behind?

What topic(s) are the consumer group consuming messages from?

What is the retention rate for my app topics?

Who are all the producers and consumers connected to my app topics?

Did a consumer rebalance occur for a given topic?

How many active consumers instances are in a given consumer group?

What is the replication factor for my app topics?

What type of events are in my application topics? What does the event look like?

Are any of my consumers/consumer-groups that are over-consuming?

Are any of my consumers/consumer-groups that are under-consuming?

# Governance Team



Governance / Security  
Team Concerns

When was a topic created?	What is the schema for a given topic?	What is the lineage of a kafka topic?
How has the schema evolved for a given topic?	How does data flow across multiple kafka hops?	Who has edited the ACL policies of a given topic?
Which consumers have consumed from a topic?	What are the ACL policies for a given topic?	When were additional brokers added to the topic?
Which users/groups/service accounts have read from a given topic?	Which users/groups/service accounts have sent data to a topic?	Which producers have sent data to the topic?
When was the topic configuration last modified?		

# Chapter Topics

---

## Monitoring Kafka

- Monitoring Overview
- **Monitoring using Cloudera Manager**
- Charts and Reports in CM
- Monitoring Recommendations
- Metrics for Troubleshooting
- Diagnosing Service Failure
- Hands-On Exercise: Monitoring Kafka
- Essential Points

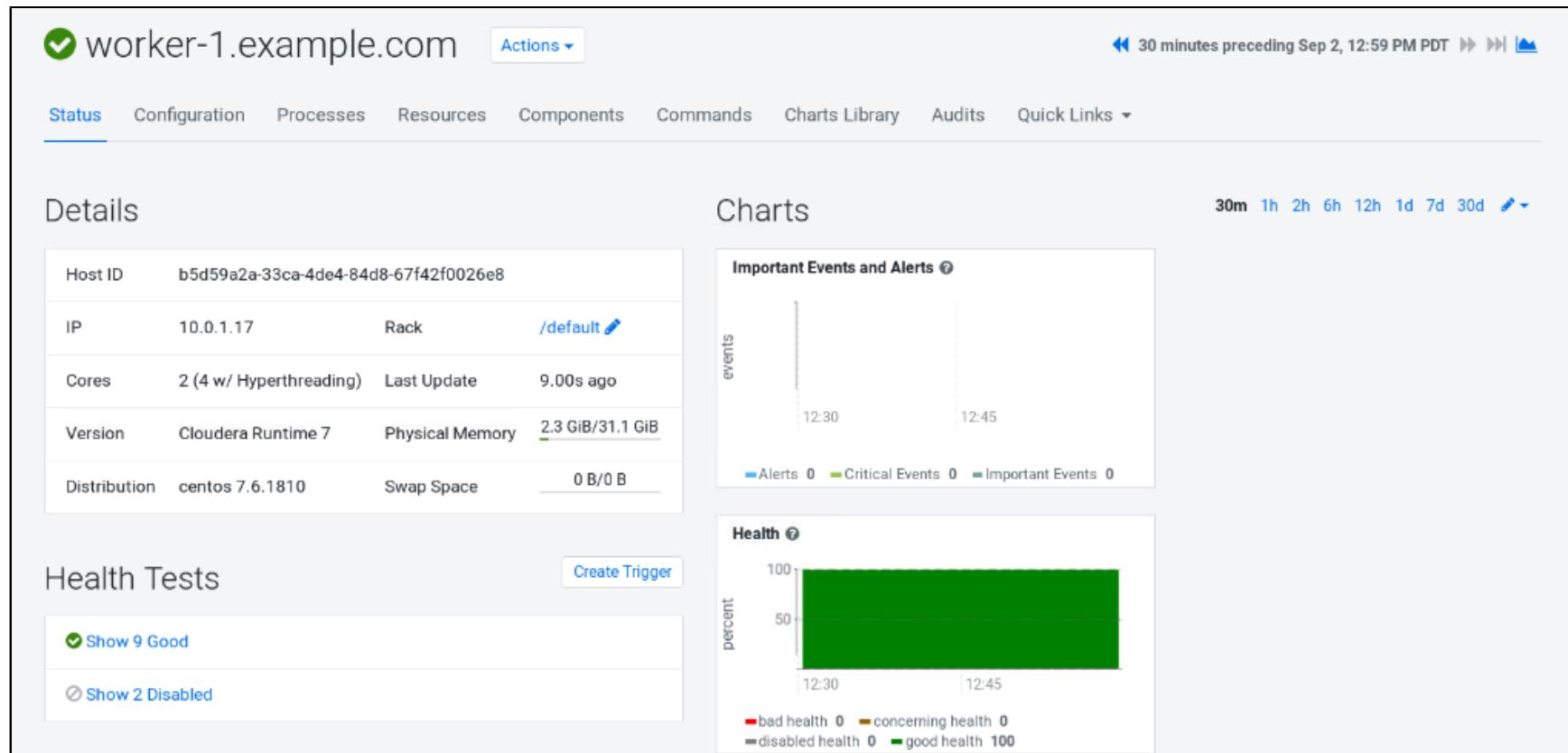
# Monitoring Terminology

---

- **Entity—a Cloudera Manager component with metrics associated with it**
  - Examples: clusters, services, roles and role instances, hosts
- **Metric—a property that can be measured**
  - Cloudera Manager monitors performance metrics cluster entities such as hosts and services
  - Examples: RAM utilization, total HDFS storage capacity
- **Chart—customizable display aggregated metrics for entities over time**
- **Dashboard—a page displaying key entity information and charts**

# Entity Status Tab

- Pages for clusters, services, hosts, roles, and other entities have Status tabs
  - Customizable dashboards with key details about the current state



# Health Tests

---

- **Cloudera Manager monitors the health of services, roles, and hosts**
- **Pass-fail tests—entity is either “good” or “bad”**
  - *Canary test:* Does service appear to be working correctly?
  - *Yes-no test:* Check for a specific property
  - Example: Are all DataNodes connected to a NameNode?
- **Metric tests—compare numeric value to a configurable *threshold***
  - Result is “good”, “bad”, or “concerning”
  - Example: does sufficient HDFS capacity remain?
    - “Concerning” threshold: < 60% remaining
    - “Bad” threshold: < 80% remaining

# Monitoring Health Issues in Cloudera Manager (1)

## ■ Cluster Health Status Indicators



Good—test result above all thresholds



Concerning—test result below the “warning” threshold



Bad—test result below the “critical” threshold

**Cluster Status**

The screenshot shows the 'Cluster Status' page. At the top, there's a navigation bar with tabs: Home (selected), Status, All Health Issues (with a red badge showing 07), Configuration, and All Recent Commands. Below the navigation, it says 'Cluster 1 (CDH)' with a warning icon. A red box highlights the 'All Health Issues' tab, and an arrow points from it to a callout box labeled 'All cluster health issues'. Another red box highlights the 'Service health issues' section, which has arrows pointing to specific service status rows: 'HDFS' (2 concerns), 'Hive' (2 concerns), 'Hue' (1 concern), 'Impala' (2 concerns), 'Kafka' (1 concern), and 'Oozie' (0 concerns).

Service	Status	Count
HDFS	!	2
Hive	!	2
Hue	!	1
Impala	!	2
Kafka	!	1
Oozie	✓	0

# Monitoring Health Issues in Cloudera Manager (2)

- You can suppress health tests or view test status, corrective actions, and advice

### Health Issues Detail

Home    Status    All Health Issues 05

Organize By Entity    Organize By Health Test

Cluster 1

- DataNode (worker-1)
  - Process Status    Suppress...
- Impala Daemon (worker-1)
  - StateStore Connectivity    Suppress...
  - Process Status    Suppress...
- Kafka Broker (worker-1)
  - Process Status    Suppress...
- NodeManager (worker-1)
  - Process Status    Suppress...

### StateStore Connectivity (Cluster 1, Impala, Impala Daemon, worker-1)

Test of whether the StateStore considers this Impala Daemon alive.

Bad : This Impala Daemon is not connected to its StateStore.

#### Actions

- Change Impala Daemon Connectivity Health Test for all roles in Impala Daemon Default Group
- Change Impala Daemon Connectivity Tolerance at Startup for all roles in Impala Daemon Default Group
- Change Health Test Startup Tolerance for all roles in Impala Daemon Default Group role group
- Change Impala Daemon Connectivity Health Test for this role instance
- Change Impala Daemon Connectivity Tolerance at Startup for this role instance
- Change Health Test Startup Tolerance for this role instance
- View the log for this role instance at the time of the health test

#### Advice

This is an Impala Daemon health test that checks whether the StateStore considers the Impala Daemon alive. A failure of this health test may indicate that the Impala Daemon is having trouble communicating with the StateStore web server. This test may return an unknown result if the Service Monitor is not able to communicate with the StateStore web server and the Service Monitor logs if this test is returning an unknown result.

This test can be enabled or disabled using the Impala Daemon Connectivity Health Test Impala Daemon

# Events

---

- An **event** is a record of something of interest that occurred
  - Default settings enable the capture of many events
- Event types include (click on + or the filter icon to add an event)
  - ACTIVITY\_EVENT—jobs that fail or run slowly
  - AUDIT\_EVENT—actions taken in Cloudera Manager such as starting a role
  - HEALTH\_CHECK—health test results
  - LOG\_MESSAGE—log messages from Kafka, ZooKeeper, etc.

Diagnostics > Events

Events

Service:  + 30m 1h 2h 6h 12h 1d 2d 30d

September 4, 2020 6:54 AM INFORMATIONAL The health test result for DATA\_NODE\_HA\_CONNECTIVITY has become disabled: Test disabled while the role is still running. This DataNode is connected to each running NameNode.

The health test result for DATA\_NODE\_TRANSCEIVERS\_USAGE has become disabled: Test disabled while the role is still running. This role has too many open transceivers.

The health test result for DATA\_NODE\_FILE\_DESCRIPTOR has become disabled: Test disabled while the role is still running. This role has too many open file descriptors.

The health test result for DATA\_NODE\_SWAP\_MEMORY\_USAGE has become disabled: Test disabled while the role is still running. This role is using swap memory.

The health test result for DATA\_NODE\_HOST\_HEALTH has become disabled: Test disabled while the role is still running. This host running this role is healthy.

The health test result for DATA\_NODE\_WEB\_METRIC\_COLLECTION has become disabled: Test disabled while the role is still running. This role's web server is responding to requests for metrics.

The health test result for DATA\_NODE\_PAUSE\_DURATION has become disabled: Test disabled while the role is still running. This role's threads are being scheduled appropriately.

> Service: HDFS Role: DataNode(worker1) Hosts: worker1.example.com Category: HEALTH\_CHECK

September 4, 2020 6:54 AM INFORMATIONAL The health test result for NODE\_MANAGER\_HEALTH\_CHECKER has become disabled: Test disabled while the role is still running. The YARN ResourceManager sees the NodeManager as healthy.

The health test result for NODE\_MANAGER\_FILE\_DESCRIPTOR has become disabled: Test disabled while the role is still running. The NodeManager sees the FileDescriptor as healthy.

# Alerts

---

- Alerts are events triggered by “noteworthy” events
- Alerts can be configured for
  - Activity running too slowly
  - A configuration was changed
  - Health condition thresholds not met on a role or host
  - Log messages or events that match a condition you define
- Alerts are noted in the Cloudera Manager event viewer

Diagnostics > Events

Events ◀ 30 minutes preceding Sep 4, 6:55 AM PDT ▶ 30d

Alert = YES Search Suggestions ▾

30m 1h 2h 6h 12h 1d 7d 30d

Time	Level	Message
September 4, 2020 6:52 AM	CRITICAL	The health test result for IMPALA_IMPALADS_HEALTHY has become bad: Healthy Impala Daemon: 1. Concerning Impala Daemon: 0. Total Impala Daemon: 2. Percent healthy: 50.00%. Percent healthy or concerning: 50.00%. Critical threshold: 90.00%.
> Service: Impala	Category: HEALTH_CHECK	
September 4, 2020 6:52 AM	CRITICAL	The health test result for IMPALAD_SCM_HEALTH has become bad: This role's process is started. This role is supposed to be stopping.
> Service: Impala	Category: HEALTH_CHECK	
September 4, 2020 6:52 AM	INFORMATIONAL	The health test result for IMPALAD_CONNECTIVITY has become disabled: Test disabled while the role is stopping: Test of whether the StateStore considers this Impala Daemon alive. The health test result for IMPALAD_READY_STATUS has become disabled: Test disabled while the role is stopping: Test of whether the Impala Daemon is ready to process queries. The health test result for IMPALAD_QUERY_MONITORING_STATUS has become disabled: Test disabled while the role is stopping: Test of whether query monitoring for the Impala Daemon is

# Alert Delivery

- Alert delivery options
  - Send email to an address you configure
  - Send SNMP\* traps for external monitoring systems
- Enable and configure event delivery in Cloudera Manager

## Cloudera Manager Service > Configuration tab OR Administration > Alerts

The screenshot shows the Cloudera Management Service configuration interface. The top navigation bar includes Status, Instances, Configuration (which is selected), Commands, Charts Library, Audits, and Quick Links. A search bar and filter buttons (Filters, Role Groups, History and Rollback) are also present. On the left, there's a sidebar with 'Filters' and sections for 'SCOPE' and 'CATEGORY'. The main area displays several alert configuration items:

- Alerts: Enable Email Alerts: Alert Publisher Default Group (checkbox checked)
- Alerts: Mail Server Protocol: Alert Publisher Default Group (radio buttons for smtp and smtps, with smtp selected)
- Alerts: Mail Server Hostname: Alert Publisher Default Group (text input field containing localhost)
- Alerts: Mail Server Username: Alert Publisher Default Group (text input field, currently empty)
- Alerts: Mail Server Password: Alert Publisher Default Group (text input field, currently empty)

\* Simple Network Management Protocol

# Viewing Enabled and Disabled Alerts

- View enabled and disabled alerts, organized by type and service

**Administration > Alerts**

Alerts

Health Alerts   Configuration Alerts   Log Event Capture   Log Event Extraction Rules   Mail Server   Alert SNMP   Custom Alert Script

Select the services and roles that will generate alerts when their health reaches the configured threshold.

Setting	Description	Value	Help
Health Alert Threshold	Cloudera Management Service > Event Server Default Group	<input checked="" type="radio"/> Bad <input type="radio"/> Concerning	Show All Descriptions <a href="#">?</a>
Alert On Transitions Out of Alerting Health	Cloudera Management Service > Event Server Default Group	<input type="checkbox"/>	<a href="#">?</a>
Enable Service Level Health Alerts	Cloudera Management Service (Service-Wide)	<input type="checkbox"/>	<a href="#">?</a>
<a href="#">Edit Identical Values</a>	Cluster 1 > HDFS (Service-Wide) Cluster 1 > Hive (Service-Wide) Cluster 1 > Hue (Service-Wide) Cluster 1 > Impala (Service-Wide) Cluster 1 > Oozie (Service-Wide) Cluster 1 > Spark (Service-Wide) Cluster 1 > Sqoop (Service-Wide)	<input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/> <input checked="" type="checkbox"/>	<a href="#">?</a>

# Configuring Alerts

- Enable and configure alerts for services on the Configuration tab
- Many alerts are enabled by default

**HDFS Configuration tab**

The screenshot shows the HDFS Configuration tab interface. At the top, there's a navigation bar with tabs for Status, Instances, Configuration (which is selected), Commands, File Browser, Charts Library, Cache Statistics, Audits, Web UI, and Quick Links. A search bar contains the word "alert". On the right, the date and time are displayed as Jan 17, 11:22 AM PST.

**Filters** section:

- SCOPE**:
  - HDFS (Service-Wide) (2)
  - Balancer (2)
  - DataNode (3)
  - Gateway (1)
  - HttpFS (2)
  - JournalNode (3)
  - NFS Gateway (3)
  - NameNode (3)
  - SecondaryNameNode (3)
  - Failover Controller (3)
- CATEGORY**:
  - All categories (0)

**Enable Configuration Change Alerts**:  HDFS (Service-Wide) ...and 10 others [Edit Individual Values](#) [Show All Descriptions](#) [?](#)

**Enable Service Level Health Alerts**:  HDFS (Service-Wide) [Edit Individual Values](#) [?](#)

**Rules to Extract Events from Log Files**: Balancer Default Group [View as JSON](#)

➤ Alert: false Rate: 1 Period: 1 Threshold: FATAL	<a href="#">-</a> <a href="#">+</a>
➤ Alert: false Rate: 0 Threshold: WARN Content: .* is deprecated. Instead, use.*	<a href="#">-</a> <a href="#">+</a>
➤ Alert: false Rate: 0 Threshold: WARN Content: .* is deprecated. Use.* instead	<a href="#">-</a> <a href="#">+</a>
➤ Alert: false Rate: 0 Exception Type: java.io.IOException	<a href="#">-</a> <a href="#">+</a>

[Feedback](#)

# Audit Events

- Audit events describe actions that occurred on a host or for a service or role
  - Lifecycle events—such as starting or stopping a role or host, installing or upgrading services, and activating parcels
  - Security events—such as login success or failure, and adding or deleting users
- Audit events include important details such as time, user, command, and IP address

Cloudera Manager Audits Page

The screenshot shows the 'Audits' page in Cloudera Manager. The left sidebar has a dark theme with white text and icons. The 'Audits' option is selected and highlighted in blue. The main content area has a light gray background. At the top, it says 'Audits' and 'No selected filters.' There is a 'Search' input field and a 'Add a filter' button. To the right, there are buttons for 'Search', 'Suggestions', and 'Download CSV'. Below these are two rows of audit log entries. Each entry includes a timestamp, command, status, and additional details like user and IP address.

Timestamp	Command	Status	Details
September 4, 2020 7:20 AM	RESTARTWAITINGFORSTALENESS	SUCCESS	Succeeded command RestartWaitingForStalenessSuccess
September 4, 2020 7:17 AM	RESTARTWAITINGFORSTALENESS	SUCCESS	Started command RestartWaitingForStalenessSuccess User: admin
September 4, 2020 7:14 AM	HOSTSBRINGUP	SUCCESS	Succeeded command HostsBringUp
September 4, 2020 7:14 AM	HOSTSBRINGUP	IN PROGRESS	Started command HostsBringUp User: admin IP Address: 10.0.12.63

# Chapter Topics

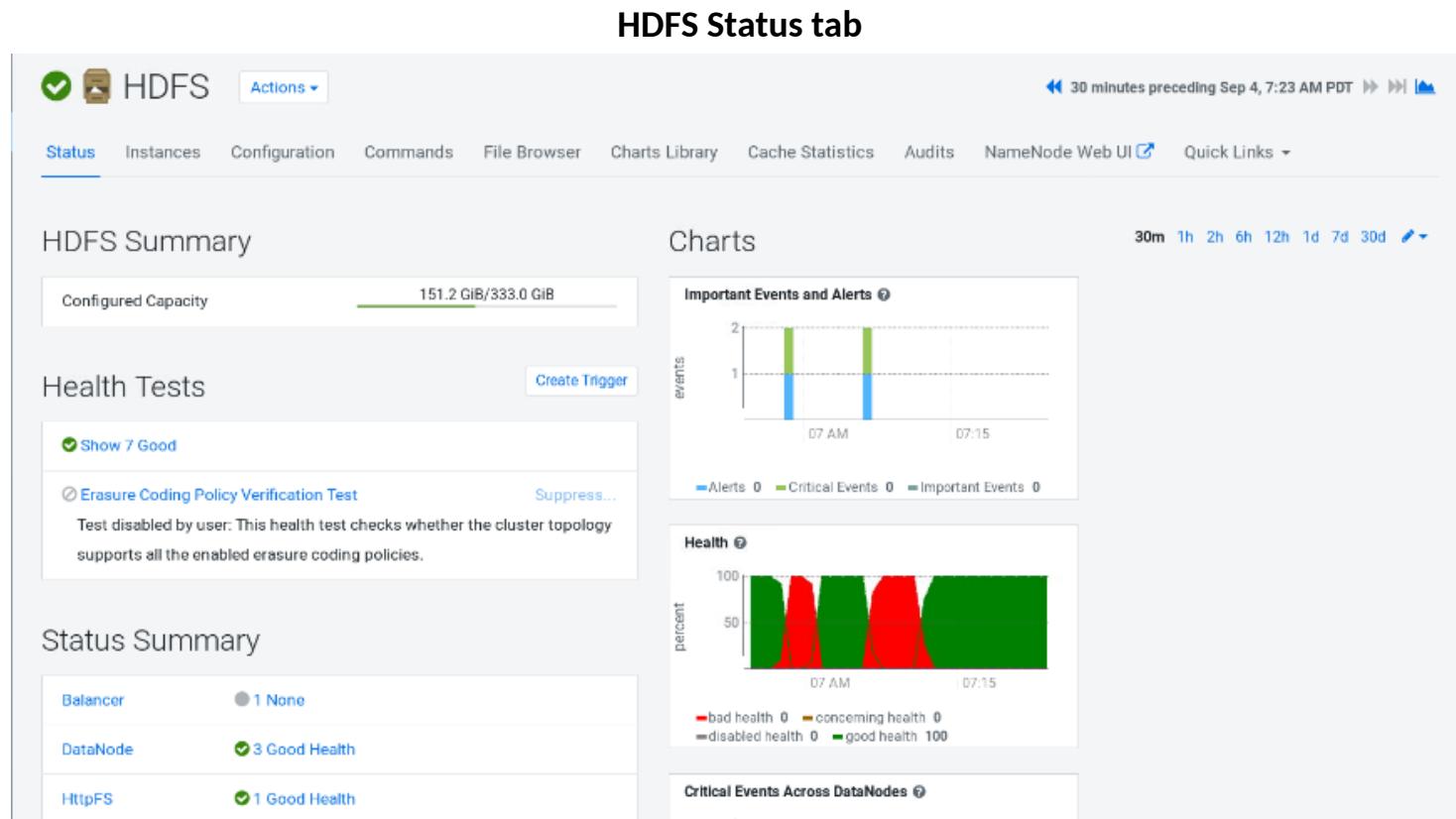
---

## Monitoring Kafka

- Monitoring Overview
- Monitoring using Cloudera Manager
- **Charts and Reports in CM**
- Monitoring Recommendations
- Metrics for Troubleshooting
- Diagnosing Service Failure
- Hands-On Exercise: Monitoring Kafka
- Essential Points

# Pre-Built Dashboards

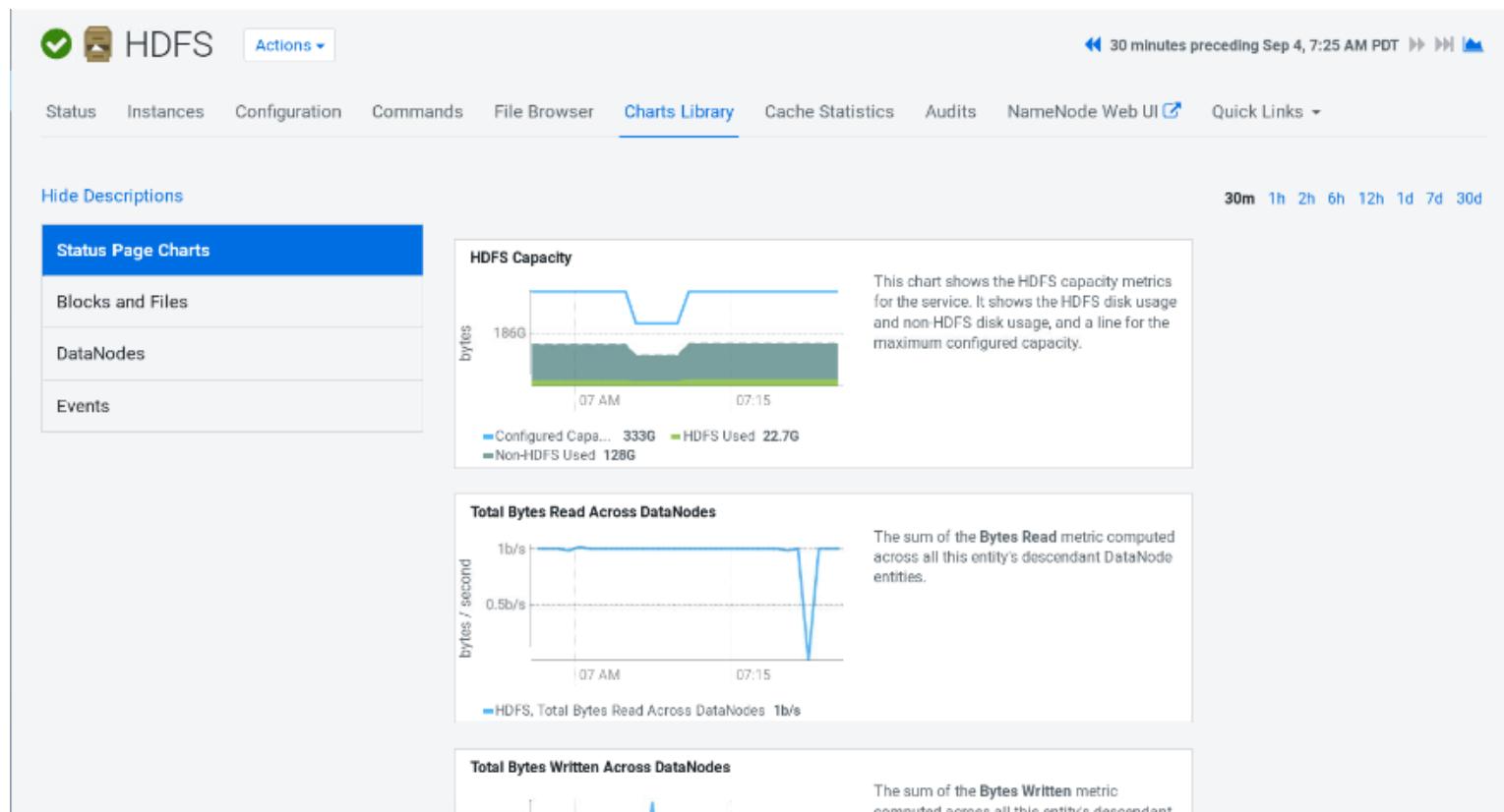
- Entity status tabs display default dashboards
  - Customizable with pre-built or custom charts



# Charts Library

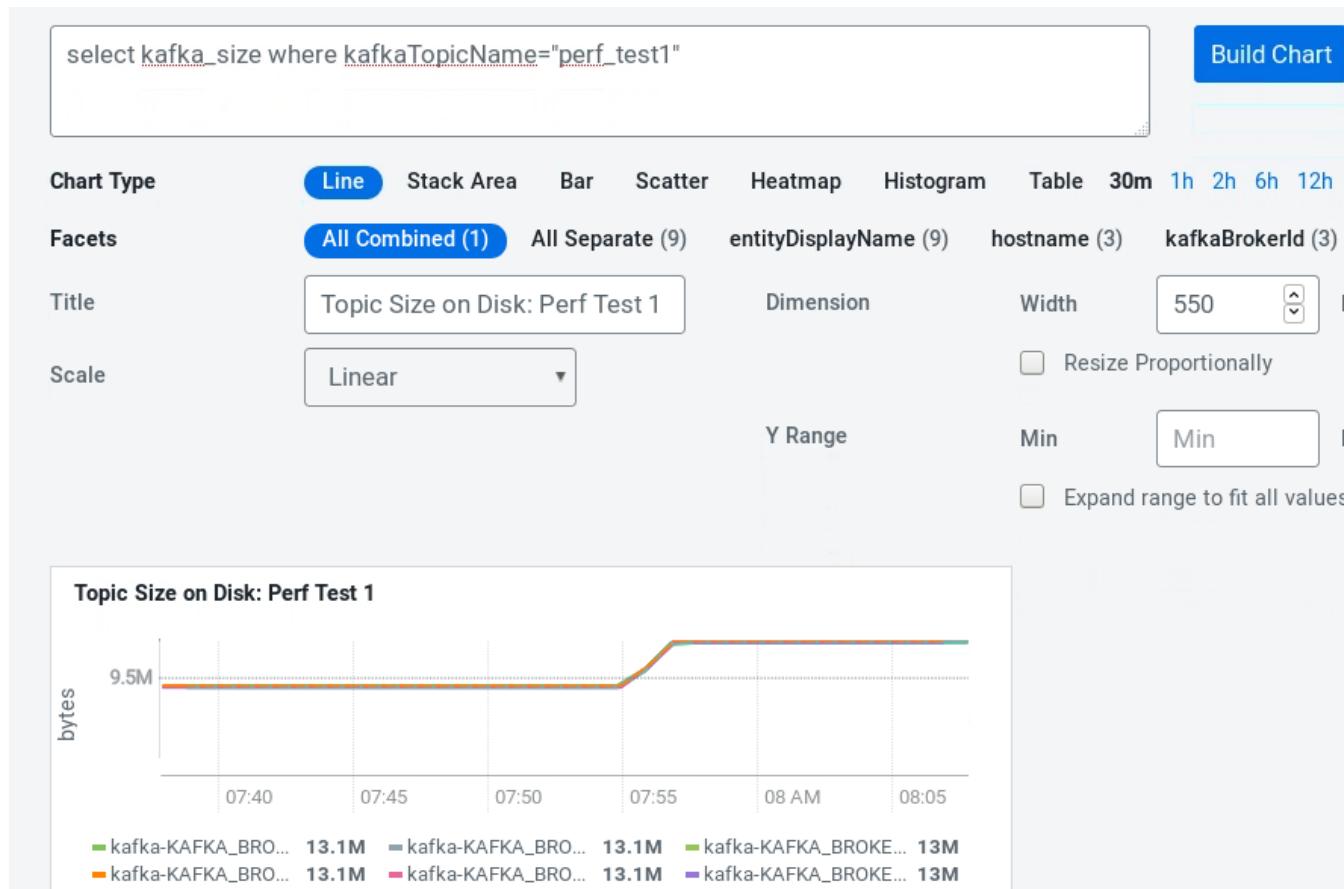
- Cloudera Manager charts library provides many pre-built charts

## HDFS Charts Library tab



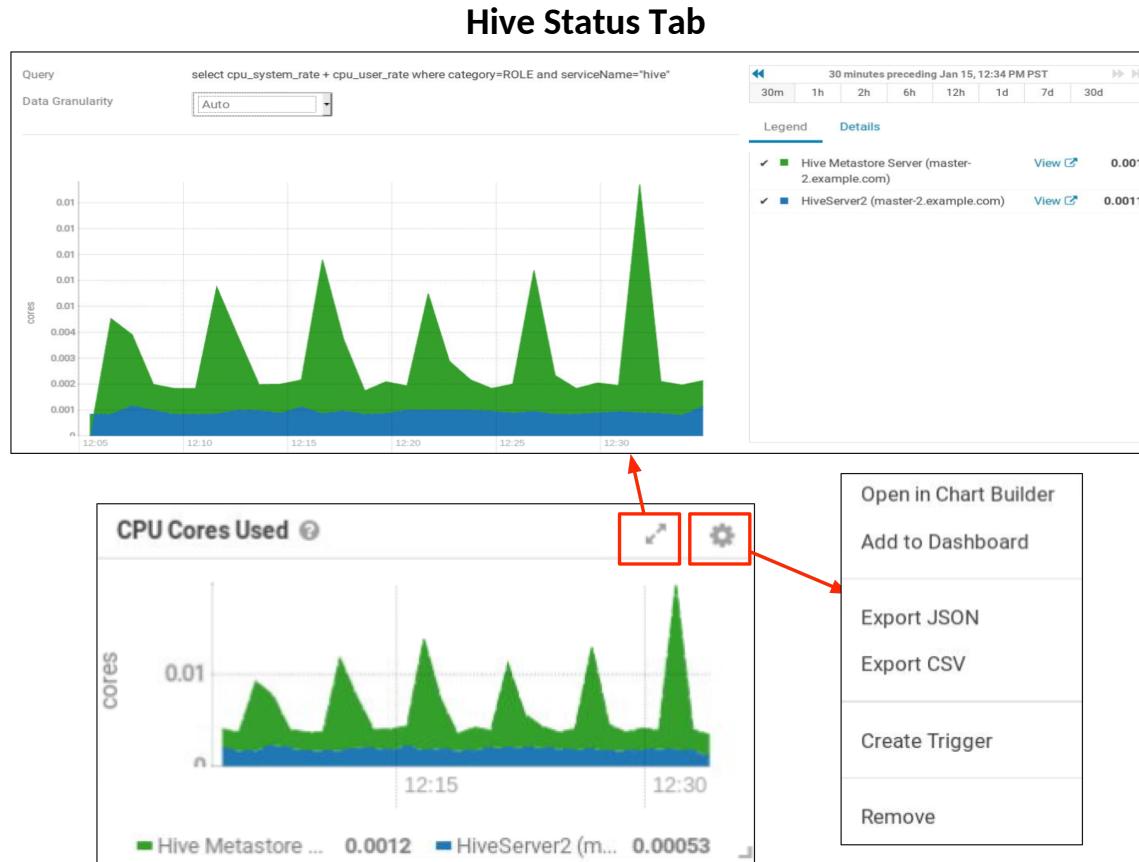
# Custom Charts

- Create custom charts to add to dashboards using the `tsquery` language
- Example: Show disk usage for a particular topic
  - `SELECT kafka_size where kafkaTopicName="events"`



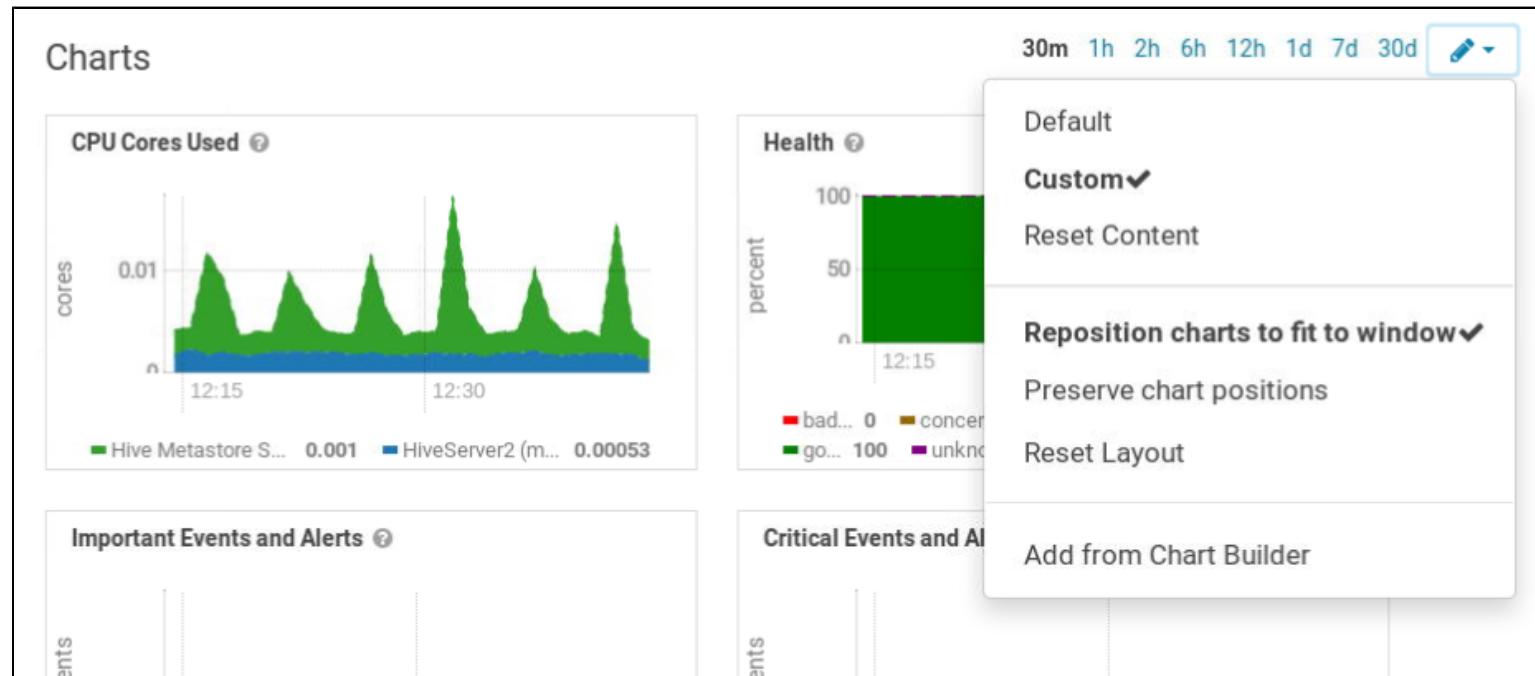
## Chart Options

- Hover over a chart to see options
    - Click on the  icon to expand the chart viewing area and details
    - Click on the  icon for more options



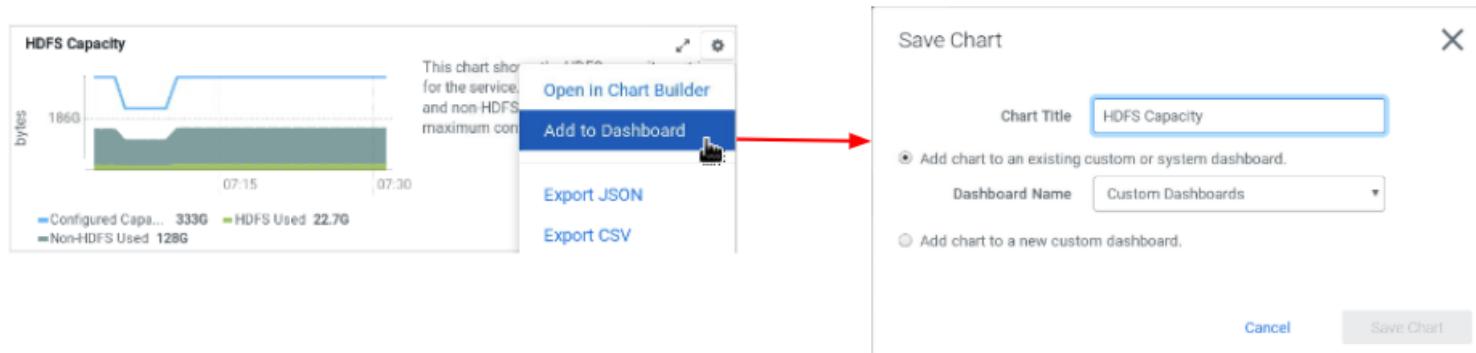
# Dashboards (1)

- A dashboard consists of a set of charts
- Cloudera provides many pre-configured dashboard
  - For example, each service's status page has a dashboard
  - Click on the  icon to manage an existing dashboard
  - Options to add or remove charts, change layout



## Dashboards (2)

- Custom dashboards add existing or custom charts

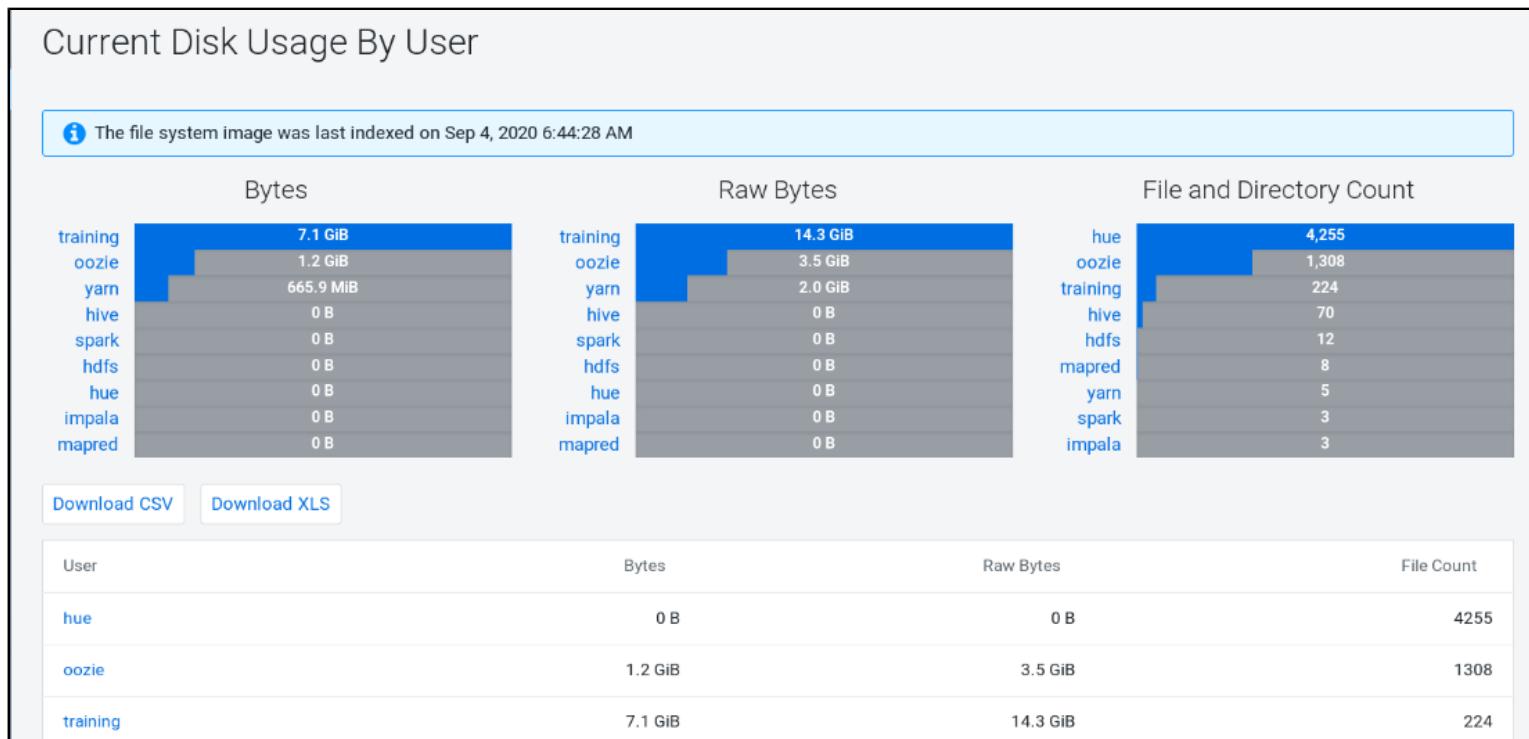


- Manage your user-defined dashboards in Charts

# Reports (YARN, Impala, HDFS, HBase)

- Per-cluster reports are available for HDFS disk usage, YARN applications, Impala queries, HDFS file access, and HBase tables and namespaces
- Download reports or view them in Cloudera Manager

Clusters > Reports > Select the Report

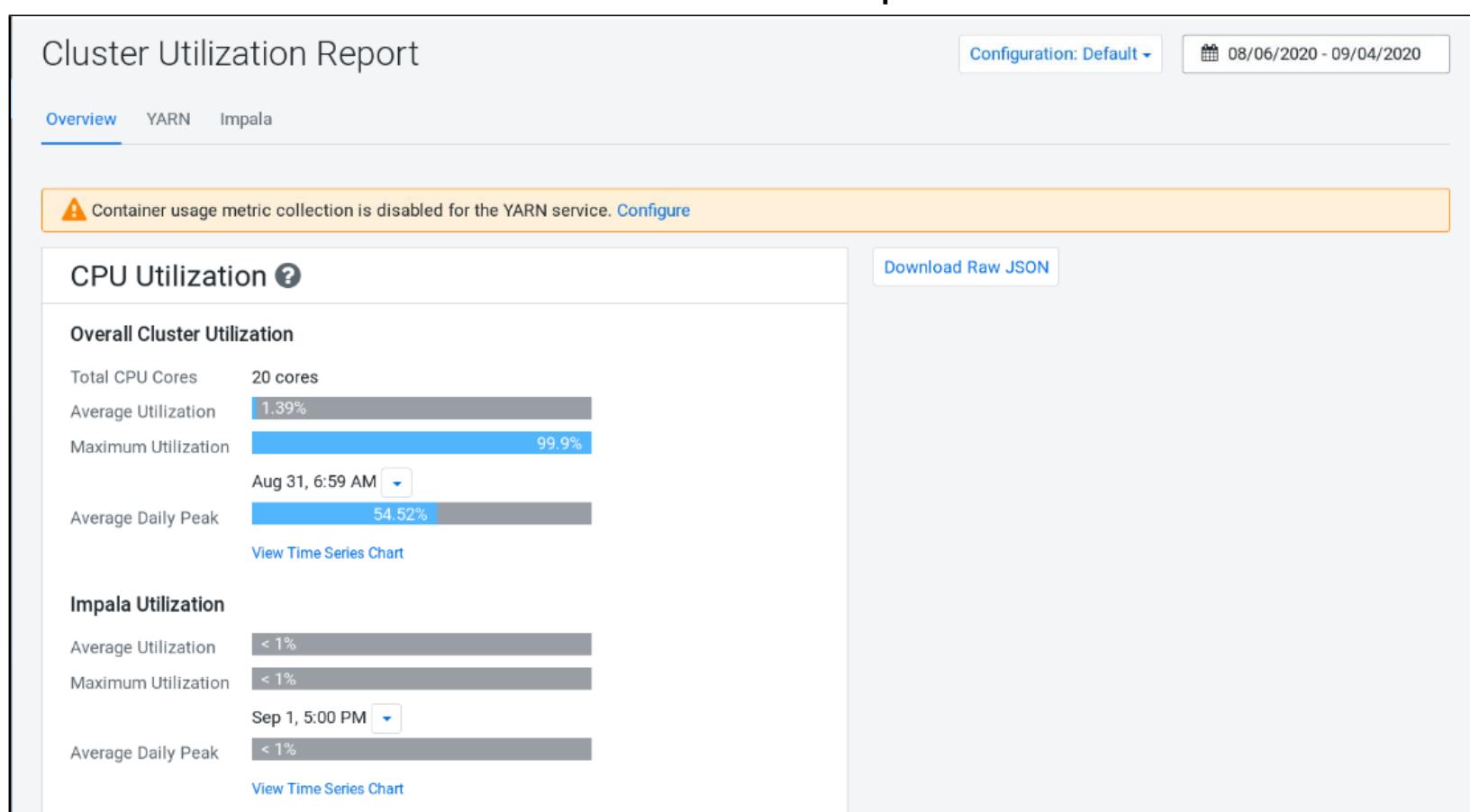


# Cluster Utilization Report (1) YARN apps and Impala

---

- The *Cluster Utilization Report* displays information about resource utilization of YARN applications and Impala queries
  - CPU utilization
  - Memory utilization
  - Resource allocation
  - Optional: YARN application metrics
    - A MapReduce job periodically aggregates metrics by application
- Metrics aggregated for a whole cluster or by *tenant* (user or resource pool)

# Cluster Utilization Report (2)



# Chapter Topics

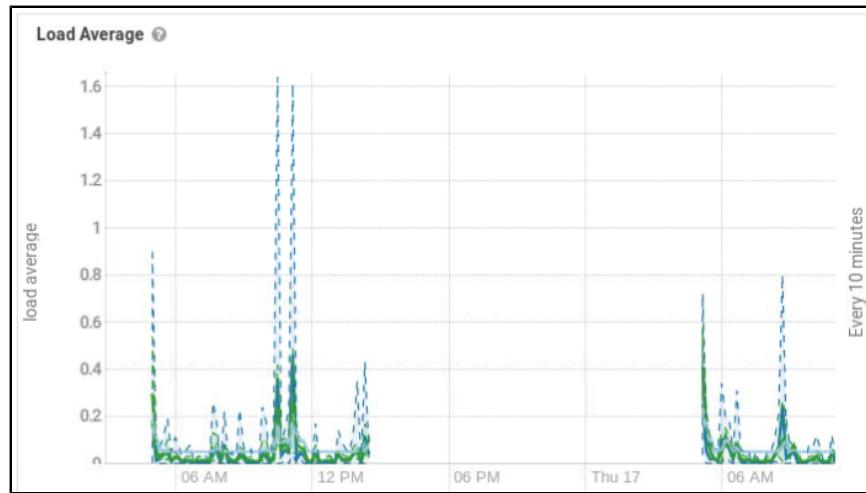
---

## Monitoring Kafka

- Monitoring Overview
- Monitoring using Cloudera Manager
- Charts and Reports in CM
- **Monitoring Recommendations**
- Metrics for Troubleshooting
- Diagnosing Service Failure
- Hands-On Exercise: Monitoring Kafka
- Essential Points

# Monitoring Recommendations: Daemons and CPU Usage

- Service daemons (e.g. Brokers)
  - Recommendation: send an alert if a daemon goes down
  - Cloudera Manager default will alert for some but not all daemons down
- CPU usage on Brokers / Zookeeper Hosts
  - Watch for excessive CPU usage and load averages for performance and utilization management
  - **Host CPU Usage**, **Roles CPU Usage**, and **Load Average** charts



# Monitoring Recommendations: Memory, Network, and Disk

---

- **Monitor network transfer speeds**
- **Monitor swapping on all hosts**
  - Warning alert if any swapping occurs—memory allocation is overcommitted
    - “Host Memory Swapping Thresholds” property
- **Monitor for disk failure and space available**

# Log File Disk Usage (1)

---

- **Kafka Broker / ZooKeeper logs**
  - Kafka Brokers and ZooKeeper daemons use Rolling File Appender, so log files are rotated
  - Configure appropriate size and retention policies in Cloudera Manager
- **YARN application logs**
  - Caution: applications by inexperienced developers will often create large container logs
  - Large task logs impact disk usage as they are written to local disks on worker hosts first
  - Ensure you have enough room locally for application logs

# Chapter Topics

---

## Monitoring Kafka

- Monitoring Overview
- Monitoring using Cloudera Manager
- Charts and Reports in CM
- Monitoring Recommendations
- **Metrics for Troubleshooting**
- Diagnosing Service Failure
- Hands-On Exercise: Monitoring Kafka
- Essential Points

# Broker Key Metrics

---

Metric	Description	Priority
Free Memory / Disk	Brokers should not be swapping. Be conservative with free disk space - brokers do not handle disks running out of space well.	Critical
Garbage Collection (GC) Pauses	High GC times will cause broker latency	High
Disk Latency	Disk latency will eventually cause message latency	Medium

## Topic Key Metrics

---

Metric	Description	Priority
Offline Partitions	Partitions for which the leader is not available	Critical
In Sync / Out of Sync Replicas	Partitions whose followers have not fetched the latest messages within <code>replica.lag.time.max.ms</code>	High
Under Replicated Partitions	Partitions which have fewer In Sync replicas than the replication factor	High

## Topic Issues:

---

- **Detection: Cloudera Manager (Streams Messaging Manager if Available)**
  - Health Tests will typically reveal topic issues
  - Lagging Replicas (not "In-Sync")
  - Offline Replicas (Follower/Leader Broker missing)
- **Diagnosis:**
  - Cloudera Manager: Use Events, Health Test Results, Log Files
  - Cloudera Manager: Use Kafka Service -> Charts -> Charts for Topic X

# Chapter Topics

---

## Monitoring Kafka

- Monitoring Overview
- Monitoring using Cloudera Manager
- Charts and Reports in CM
- Monitoring Recommendations
- Metrics for Troubleshooting
- **Diagnosing Service Failure**
- Hands-On Exercise: Monitoring Kafka
- Essential Points

# Cloudera Manager Host Inspector

- The Host Inspector gathers information about host
  - Such as HDFS settings, CDP Runtime component versions, system time, and networking configuration

## Hosts > All Hosts > Host Inspector

The screenshot shows a modal window titled "Inspect All Hosts". At the top, it displays "Status: Finished" with a timestamp of "Sep 4, 7:44:51 AM" and a duration of "10.59s". There are "Download" and "Show Inspector Results" buttons. Below this, a message says "Command completed with 5/5 successful subcommands". A section titled "Completed 5 of 5 step(s)." is expanded, showing five entries. Each entry includes a green checkmark icon, a description of the step ("Runs the host inspector on a single host."), the host name (e.g., "cmhost.example.com"), the timestamp ("Sep 4, 7:44:51 AM"), and the duration ("6.3s"). At the bottom of the modal are "Download", "Show Inspector Results", and "Close" buttons.

Description	Host	Timestamp	Duration
> ✓ Runs the host inspector on a single host.	cmhost.example.com	Sep 4, 7:44:51 AM	6.3s
> ✓ Runs the host inspector on a single host.	worker-2.example.com	Sep 4, 7:44:51 AM	10.45s
> ✓ Runs the host inspector on a single host.	worker-1.example.com	Sep 4, 7:44:51 AM	10.31s
> ✓ Runs the host inspector on a single host.	master-2.example.com	Sep 4, 7:44:51 AM	10.55s
> ✓ Runs the host inspector on a single host.	master-1.example.com	Sep 4, 7:44:51 AM	7.01s

# Accessing Service Logs

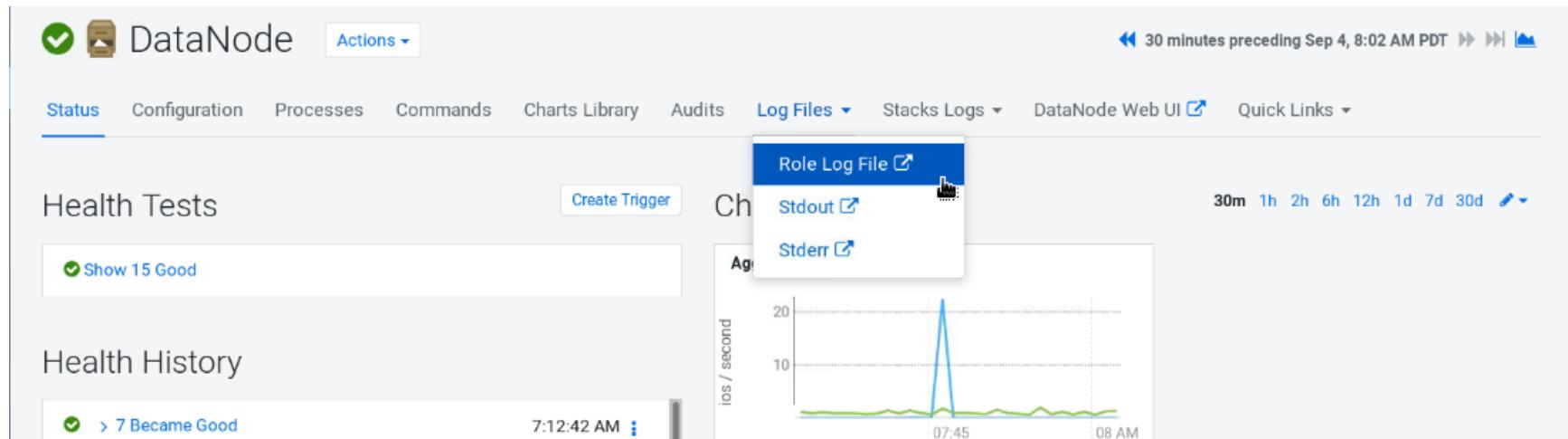
- **Diagnostics > Logs > Specify Service**
- **Log search page**
  - Filter criteria - service, role, host, and log level
  - Good for correlating across hosts or roles

The screenshot shows the Cloudera Log Search interface. The search bar at the top contains the keyword "disconnected". The "Sources" section has "Services" checked and "Cloudera Manager Agent" and "Cloudera Manager Server" unchecked. The time range is set to "2 hours preceding Feb 18, 8:27 AM PST". Below the search bar, the "Services" field is set to "Kafka". The "Minimum Log Level" is set to "WARN" and the "Timeout (sec)" is set to "60". A "Search" button is visible. The results summary below the search bar indicates "3 Machine(s) Searched (44ms). 0 Error(s), More Statistics". The main table displays log entries from "worker-2.example.com" on February 18, 2021, at 6:46 AM, with the source being "ReplicaFetcherThread". The message details a connection exception to a Kafka broker.

Hosts	Log Level	Time	Source	Message
worker-2.example.com	WARN	February 18, 2021 6:46 AM	ReplicaFetcherThread	[ReplicaFetcher replicaId=24, leaderId=23, fetcherId=replicaId=24, maxWait=500, minBytes=1, maxBytes=10485 metadata=(sessionId=1220555450, epoch=4720), rackId=) java.io.IOException: Connection to 23 was disconnected at org.apache.kafka.clients.NetworkClientUtil at kafka.server.ReplicaFetcherBlockingSend.se at kafka.server.ReplicaFetcherThread.fetchFro at

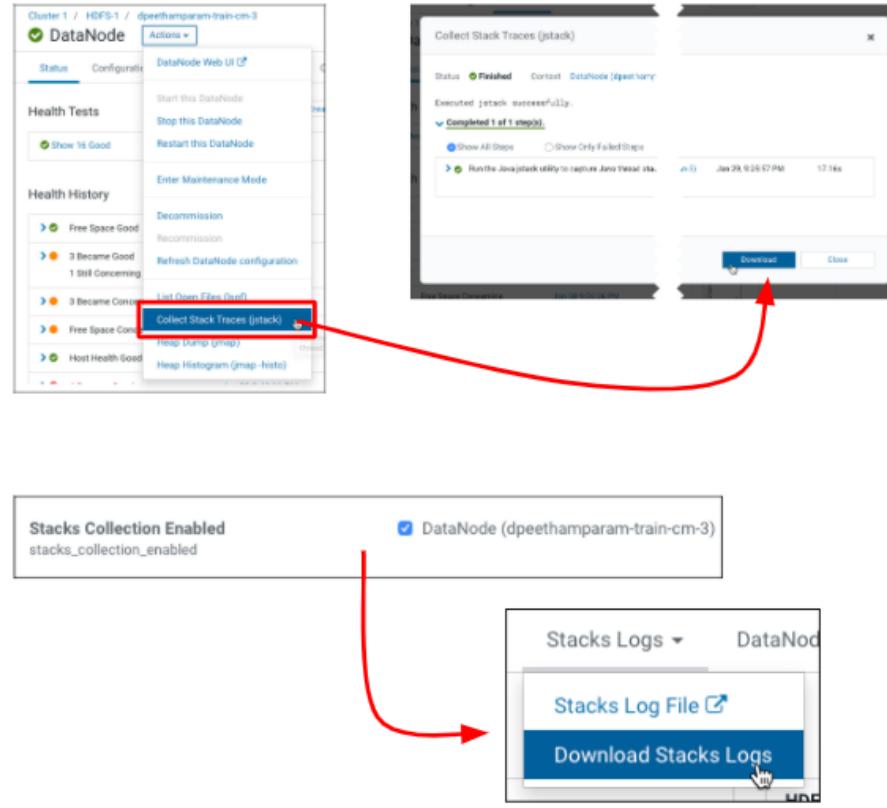
# Accessing Role Logs

- **Role instance page (e.g. Broker)**
  - Specific to that one role instance
  - Download the entire log



# Diagnostics - Thread Dump

- **Thread dump / stack trace**
  - Function call stack of each thread
  - High CPU / hung / slow response
  - Series of thread dumps
- **Single thread dump**
  - Actions menu of entity
- **Continual dump**
  - Configuration menu - Stacks Collection
  - Set the frequency or retention size
- **Retrieve from Stacks Logs menu**



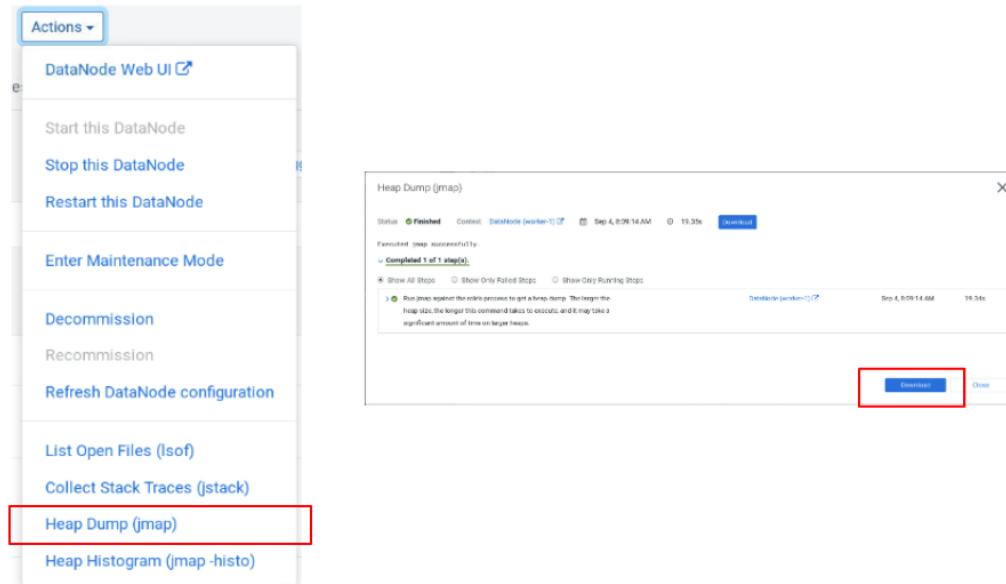
## Diagnostics - Heap Dump / Heap Histogram

---

- **Heap dump - information about objects on JVM heap**
- **Heap histogram - summary of objects**
- **Single heap dump from Actions menu**
- **Cautions: Involve Cloudera support person**
  - Causes stop the world garbage collection
  - Performance hit with large files
  - Scratch dir is /tmp (tempfs)

# Diagnostic Commands for Roles

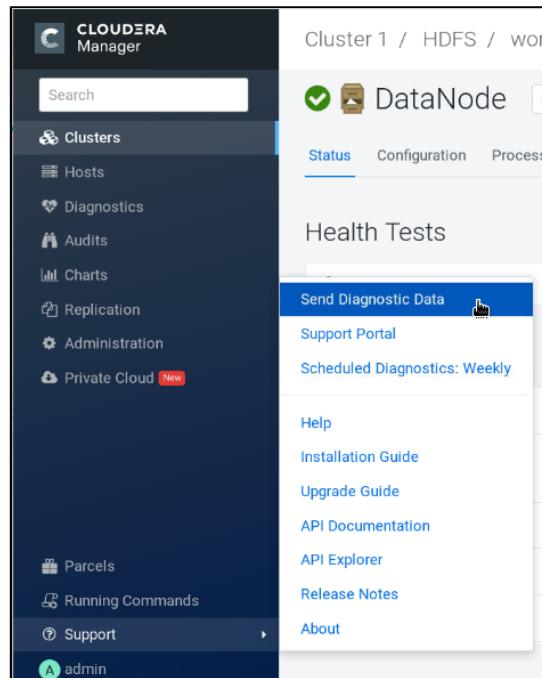
- Cloudera Manager allows administrators to run diagnostic utility tools against most Java-based role processes:
  - List Open Files (lsof) - Lists the open files of the process
  - Collect Stack Traces (jstack) - Captures Java thread stack traces
  - Heap Dump (jmap) - Captures a heap dump for the process
  - Heap Histogram (jmap -histo) - Produces a histogram of the heap
- These commands are found on the Actions menu of the Cloudera Manager page for the instance of the role



# Diagnostic Bundle

---

- Also known As "support bundle" or "diagnostic bundle"
- Gathers events, logs, config, host information, host inspector output, cluster metadata and more
- Support, Send Diagnostic Data
- Two options: collect and sendcollect only



# Setting Daemon Log Levels

- Set logging thresholds (levels) in Cloudera Manager

Hive > Configuration

The screenshot shows the 'Hive > Configuration' page in Cloudera Manager. The top navigation bar includes 'Actions ▾', 'Status', 'Instances', 'Configuration' (which is selected), 'Commands', 'Charts Library', 'Audits', 'HiveServer2 Web UI', and 'Quick Links ▾'. A search bar contains the text 'logging threshold'. Below the search bar are 'Filters' (with 'SCOPE' and 'CATEGORY' sections) and 'Role Groups' (with 'History and Rollback'). The main configuration area lists several logging thresholds:

Setting	Default Group	Value
Gateway Logging Threshold	Gateway Default Group	INFO
Hive Metastore Server Logging Threshold	Hive Metastore Server Default Group ...and 1 other	INFO
HiveServer2 Logging Threshold	HiveServer2 Default Group	INFO
WebHCat Server Logging Threshold	WebHCat Server Default Group	INFO

A 'Save Changes (CTRL+S)' button is located at the bottom right.

# Finding Consumers and Producers Settings

---

- Producers and consumers using the Java Kafka API (typically) display their configuration settings
  - To stdout and/or to their log files
  - Useful for troubleshooting

```
$ kafka-producer-perf-test --bootstrap-server ... --topic customers --producer-props  
client.id=TEST_CLIENT acks=all ...
```

```
INFO producer.ProducerConfig: ProducerConfig values:  
  acks = all  
  client.id = TEST_CLIENT  
  batch.size = 16384  
  bootstrap.servers = [worker-1.example.com:9092, worker-2.example.com:9092,  
master-2.example.com:9092]  
  buffer.memory = 33554432  
  delivery.timeout.ms = 120000  
  ...
```

# Finding Brokers Settings

---

- **Settings are usually printed to `stdout` or log files**
- - Cloudera Manager's "Configuration" page is usually best for viewing brokers' settings
  - Brokers' effective settings are also shown in brokers' log files (Search using CM->Logs, or directly from filesystem on broker)

```
2020-11-02 06:44:56,849 INFO kafka.server.KafkaConfig:  
KafkaConfig values:  
    auto.leader.rebalance.enable = true  
    background.threads = 10  
    broker.id = 149  
    broker.rack = null  
    default.replication.factor = 1  
    log.dirs = /var/local/kafka/data  
    min.insync.replicas = 1  
    num.replica.fetchers = 4  
    port = 9092  
    replica.lag.time.max.ms = 10000  
    unclean.leader.election.enable = false  
    ...
```



# Chapter Topics

---

## Monitoring Kafka

- Monitoring Overview
- Monitoring using Cloudera Manager
- Charts and Reports in CM
- Monitoring Recommendations
- Metrics for Troubleshooting
- Diagnosing Service Failure
- **Hands-On Exercise: Monitoring Kafka**
- Essential Points

## Hands-On Exercise: Monitoring Kafka

---

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

# Chapter Topics

---

## Monitoring Kafka

- Monitoring Overview
- Monitoring using Cloudera Manager
- Charts and Reports in CM
- Monitoring Recommendations
- Metrics for Troubleshooting
- Diagnosing Service Failure
- Hands-On Exercise: Monitoring Kafka
- **Essential Points**

## Essential Points

---

- **Use Cloudera Manager to visualize Kafka resources and services**
  - Broker and topic metrics
  - Charts
- **Kafka logs are the first line of investigation for Kafka issues**

# Bibliography

---

The following offer more information on topics discussed in this chapter

- Essential metrics to monitor



# Managing Kafka

---

Chapter 12

# Course Chapters

---

- Introduction
- Kafka Overview
- Deploying Apache Kafka
- Kafka Command Line Basics
- Using Streams Messaging Manager (SMM)
- Kafka Java API Basics
- Improving Availability through Replication
- Improving Application Scalability
- Improving Application Reliability
- Analyzing Kafka Clusters with SMM
- Monitoring Kafka
- Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka Cluster
- Conclusion

# Managing Kafka

---

By the end of this chapter, you will be able to

- **Manage Kafka topic storage**
- **Rebalance partitions**
- **Manage Kafka using Cruise Control**
- **Perform an unclean leader election**
- **Add and remove brokers**

# Chapter Topics

---

## Managing Kafka

- **Managing Kafka Topic Storage**
- Optional Instructor-Led Demonstration: Message Retention Period
- Log Cleanup and Collection
- Rebalancing Partitions
- Cruise Control
- Hands-On Exercise: Installing Cruise Control
- Hands-On Exercise: Troubleshooting Kafka Topics
- Unclean Leader Election
- Hands-On Exercise: Unclean Leader Election
- Adding and Removing Brokers
- Hands-On Exercise: Adding and Removing Brokers
- Best Practices
- Essential Points

# Broker Log Management

- **Messages are saved in files in log files Linux directory**
  - Continuous record of Kafka messages stored in fixed-size Linux files
  - Not to be confused with server application logs
  - Contains a list of Linux directories to support multiple disk mount points
  - Kafka Topic messages are not immediately removed after they are consumed

The screenshot shows the Cloudera Manager configuration interface for Kafka brokers. The search bar at the top contains "log dirs". Below it, there are tabs for "Filters", "Role Groups", and "History and Rollback". On the left, a sidebar has a search bar for "log dirs" and sections for "Filters" (with "SCOPE" and "CATEGORY" dropdowns) and "Logs" (with "Main", "Monitoring", "Performance", and "Advanced" options). The main panel displays configuration settings:

- Data Directories:** Set to "/var/local/kafka/data". A tooltip explains: "A list of one or more directories in which Kafka data is stored. Each new partition created is placed in the directory that currently has the least amount of partitions. Each directory should be on its own separate drive."
- Number of Alter Log Dir Threads:** Set to "num.replica.alter.log.dirs.threads".
- Number of Recovery Threads per data directory:** Set to "num.recovery.threads.per.data.dir".

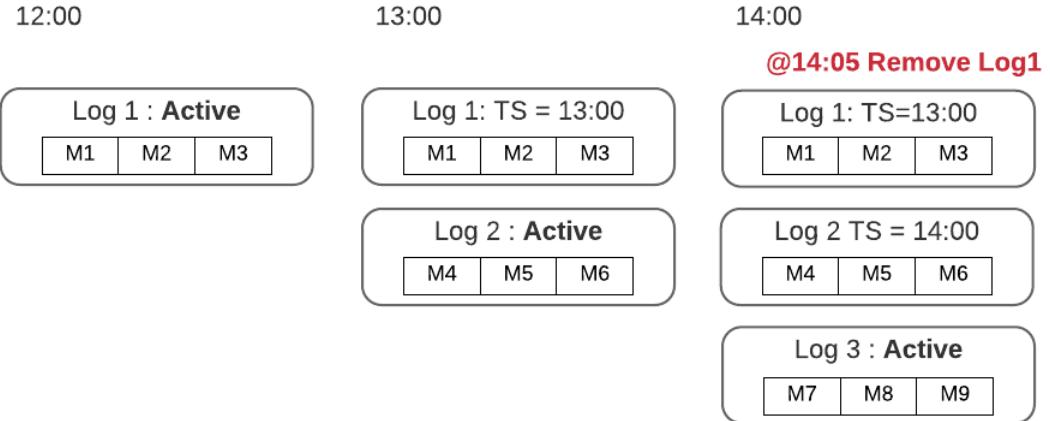
# Message Retention

---

- **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

---



# Kafka Log Size and Retention

- Kafka rolls logs periodically based on size or time limits
  - `log.retention.bytes` -1 (default) stores all messages received disk space permitting
  - `log.retention.ms` 7 days (default) deletes old messages automatically
  - Possible to set `retention.ms` to -1 and then all messages are stored
  - Older log files are deleted when max message log file size is reached

The screenshot shows the Cloudera Manager interface for managing log retention settings. The search bar at the top contains the text "log retention". Below the search bar, there are tabs for "Filters", "Role Groups", and "History and Rollback". On the left, a sidebar titled "Filters" includes sections for "SCOPE", "CATEGORY", and "STATUS". Under "SCOPE", "Kafka (Service-Wide)" is selected. Under "CATEGORY", "Main" is selected. Under "STATUS", all items are listed with a value of 0. The main content area displays three configuration items:

- Log Compaction Delete Record Retention Time**: Set to "Kafka (Service-Wide)" and "log.cleaner.delete.retention.ms" with a value of 7 days.
- Data Retention Time**: Set to "Kafka Broker Default Group" and "log.retention.ms" with a value of 7 days.
- Data Retention Size**: Set to "Kafka Broker Default Group" and "log.retention.bytes" with a value of -1 B.

Each configuration item has a detailed description to its right:

- Log Compaction Delete Record Retention Time**: Describes the amount of time to retain delete messages for log compacted topics.
- Data Retention Time**: Describes the maximum time before a new log segment is rolled out.
- Data Retention Size**: Describes the amount of data to retain in the log for each topic-partition.

# Kafka Record Management

---

- Messages are saved in files in `log` files Linux directory
- Kafka messages are stored in Linux files grouping messages in fixed sized log segments
- Log segments help protect the data from corruption due to excessive sized log files
- Topic > Partition > Log file > Log segment
  - `log.roll.ms` or `log.roll.hours`: The time period for each log segment
  - `log.segment.bytes`: The maximum size for a single log segment
- Segments larger or older than the values trigger a new log segment rotation

# Chapter Topics

---

## Managing Kafka

- Managing Kafka Topic Storage
- **Optional Instructor-Led Demonstration: Message Retention Period**
- Log Cleanup and Collection
- Rebalancing Partitions
- Cruise Control
- Hands-On Exercise: Installing Cruise Control
- Hands-On Exercise: Troubleshooting Kafka Topics
- Unclean Leader Election
- Hands-On Exercise: Unclean Leader Election
- Adding and Removing Brokers
- Hands-On Exercise: Adding and Removing Brokers
- Best Practices
- Essential Points

## Optional Instructor-Led Demonstration: Message Retention Period

---

- Your instructor will now demonstrate the effect of the message retention period
  - This demo will roll log files every 30 seconds and remove log files older than 60 seconds
  - See "Demonstrating the Effect of the Message Retention Period" in the "Demos" section of the Exercise Guide

# Chapter Topics

---

## Managing Kafka

- Managing Kafka Topic Storage
- Optional Instructor-Led Demonstration: Message Retention Period
- **Log Cleanup and Collection**
- Rebalancing Partitions
- Cruise Control
- Hands-On Exercise: Installing Cruise Control
- Hands-On Exercise: Troubleshooting Kafka Topics
- Unclean Leader Election
- Hands-On Exercise: Unclean Leader Election
- Adding and Removing Brokers
- Hands-On Exercise: Adding and Removing Brokers
- Best Practices
- Essential Points

# Log Cleanup / Compaction

---

- **Cleanup is the concept of making data expire**
  - For example: duplicate messages or tombstone delete markers
- **Cleanup is fine grain alternative to simply deleting log segments for a partition**
- **Controlled when creating individual topic by specific topic configuration**
  - Settings are: **Delete** (default) or **Compact** (enable cleanup)
- **Why do log cleanup?**
  - Optimize by restricting log segments growth size
  - GDPR compliance
    - Delete personal info stored Kafka topics
  - Good for fast message caching scenarios
    - Keep the latest value for each record in near real time

# Log Cleanup Example

---

- Topic created with `cleanup.policy = compact`
- Replacement log files are created by scanning current log files
- Older keys or tombstone delete markers cause messages to be removed

Before cleanup:

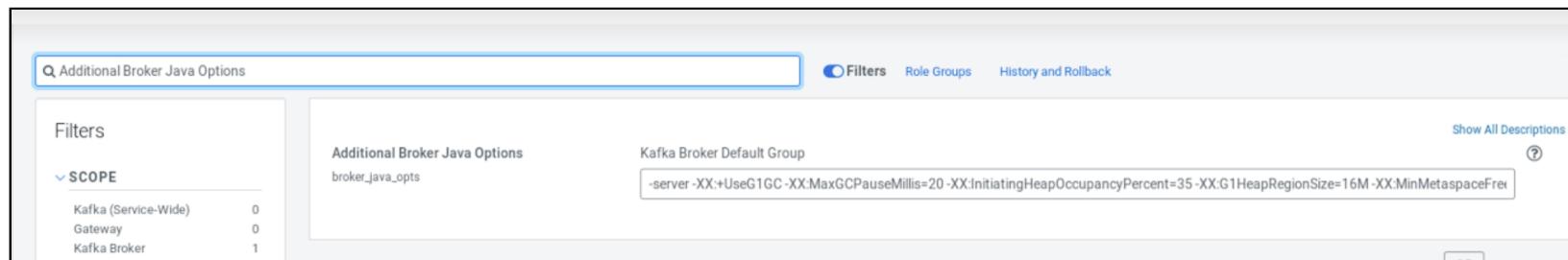
offset	1	2	3	4
key	k8	k2	k2	k14
value	\$9	\$8	\$7	\$4

After cleanup compaction:

offset	1	3	4
key	k8	k2	k14
value	\$9	\$7	\$4

# Broker Java Garbage Collection

- Java memory garbage collection has a performance impact on heavily loaded brokers
  - Large messages can cause longer garbage collection (GC) pauses as brokers allocate large chunks
  - Found under property Additional Broker Java Options
  - Add to Additional Broker Java Options to monitor the Broker GC server log
    - -XX:+PrintGC -XX:+PrintGCDetails
    - -XX:+PrintGCTimeStamps
    - -Xloggc:</path/to/file.txt>



# Key Kafka Settings for Optimal Tuning

---

- Cloudera default settings are good starting point for most clusters
- Optimize your Kafka cluster based on the number of producers, consumers and replica fetchers
- **message.max.bytes**
  - Maximum size of the message 1MB server can receive
  - Prevents any producer from inadvertently sending extra large messages
- **num.network.threads**
  - Number of threads used for handling network requests
  - Set based on number of producers, consumers and replica fetchers
  - Cluster wide setting

# Chapter Topics

---

## Managing Kafka

- Managing Kafka Topic Storage
- Optional Instructor-Led Demonstration: Message Retention Period
- Log Cleanup and Collection
- **Rebalancing Partitions**
- Cruise Control
- Hands-On Exercise: Installing Cruise Control
- Hands-On Exercise: Troubleshooting Kafka Topics
- Unclean Leader Election
- Hands-On Exercise: Unclean Leader Election
- Adding and Removing Brokers
- Hands-On Exercise: Adding and Removing Brokers
- Best Practices
- Essential Points

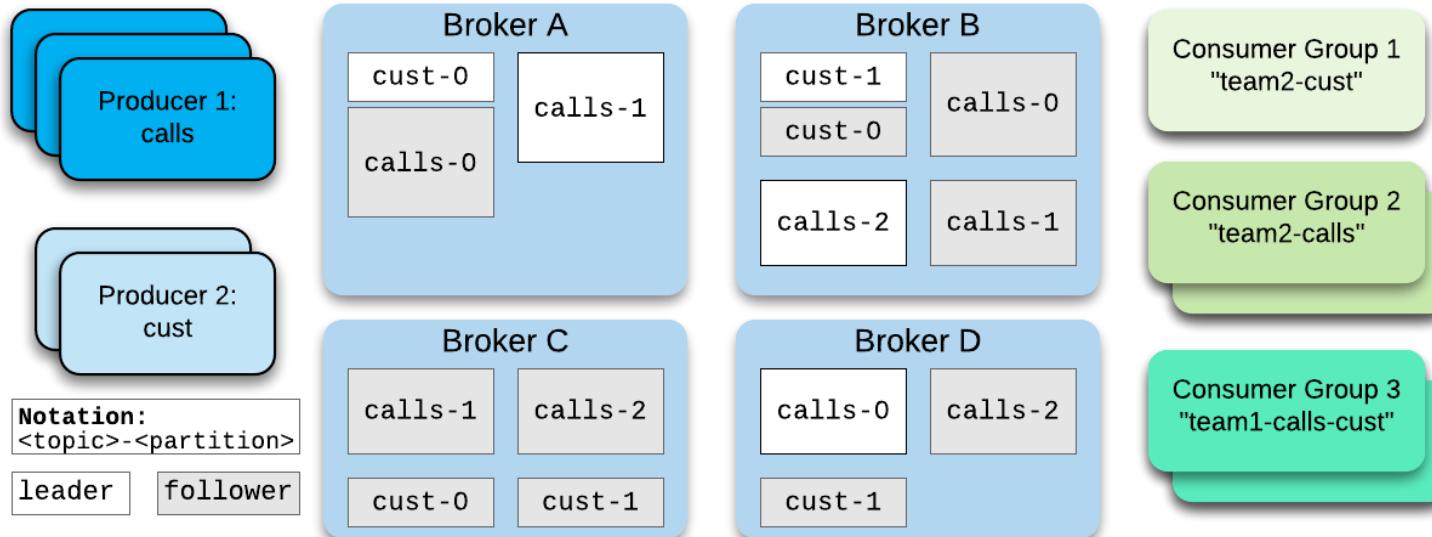
# Rebalancing Partitions

---

- **Common practice:**

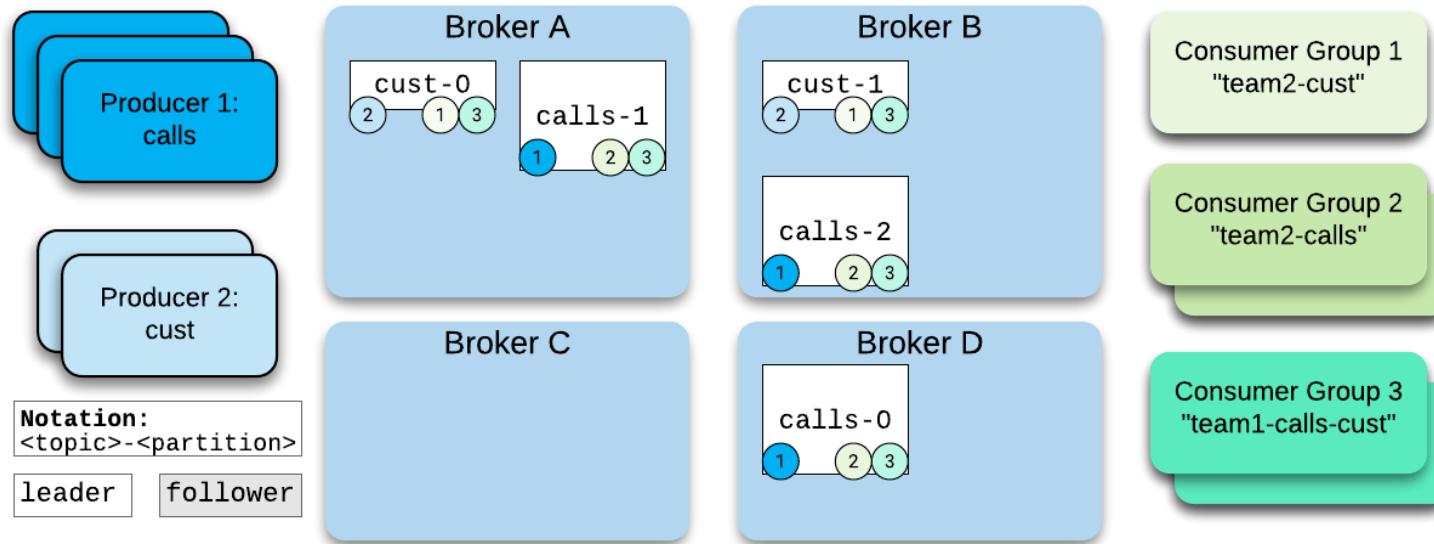
- Kafka clusters often become out of balance with regards to partition assignment
  - Brokers are assigned partitions round-robin by the Kafka controller when topics are created
  - As topics and brokers are created / removed, and producers / consumers added, resource usage of brokers can become unbalanced

# Example Partition Distribution and Activity



## Example Partition Distribution and Activity (2)

- Leader assignments are imbalanced, resulting in higher traffic to A, B and D
  - Causes: Broker removal/failure, general imbalance in topic activity



# Other Factors Affecting Optimal Partition Assignment

---

- **Distribution of partitions across racks**
  - Rack 1 has 20 partitions
  - Rack 2 has 40 partitions
- **Total number of partitions assigned to each broker**
  - Broker A may have been down when a lot of topics or a hot topic was created
  - High-activity partitions are over represented on specific brokers
- **Activity on partitions e.g. low volume on topic A and high volume on topic B**
- **Number of producers / consumers for a particular topic (imbalance of number of requests to the leader)**

# Built-in Kafka Balancer

---

- **kafka-reassign-partitions tool**
  - Uses relatively simple formula to provide suggestions for partition reassignment
  - Users may choose to reassign partitions for specific topics, specific partitions
  - Users may manually modify these suggestions
  - *Kafka Controller* receives suggestions and schedules partition reassignment
- **kafka-reassign-partitions tool may be used for other purposes**
  - Increasing replication factor for a topic
  - Reassigning leader(s) for a topic
  - "Decommissioning" brokers (e.g. moving all partition assignments away from a broker)
  - Spreading partition assignments after addition of brokers
  - Relocating storage of partitions from one disk to another
- **Reference:** [CDP 7.1.3: Reassignment Examples](#)

## Balancing Partitions, Best Practices

---

- Choose topics and partitions to balance conservatively
- Choose a low-usage time to perform rebalances
- Topics / brokers must be in a healthy state in order to perform rebalances
- Reference: [CDP 7.1.3: Managing Kafka Topics](#)

## Built-in Kafka Balancer: Issues

---

- **Uses relatively simple formula to provide suggestions**
- **Users may schedule too many partition reassessments at a time, resulting in high network traffic and broker activity**

# Chapter Topics

---

## Managing Kafka

- Managing Kafka Topic Storage
- Optional Instructor-Led Demonstration: Message Retention Period
- Log Cleanup and Collection
- Rebalancing Partitions
- **Cruise Control**
- Hands-On Exercise: Installing Cruise Control
- Hands-On Exercise: Troubleshooting Kafka Topics
- Unclean Leader Election
- Hands-On Exercise: Unclean Leader Election
- Adding and Removing Brokers
- Hands-On Exercise: Adding and Removing Brokers
- Best Practices
- Essential Points

## Another Option for Rebalancing: Cruise Control

---

- Available with CDP 7.x
- Open source project started by LinkedIn
- Provides enhanced rebalancing for Kafka Clusters
- How Cruise Control rebalancing works

## Cruise Control: "Goals"

---

- Cruise control relies on recent Kafka load information provided by Cloudera Manager
- Uses load information against pre-defined *Goals* to create a plan for rebalancing
- Examples of Goals:
  - Rack Awareness - ensures that partition replicas are assigned in a rack aware manner
  - Disk Capacity
  - Replica Distribution - An attempt to assign similar number of replicas to each broker
  - Network Inbound / Outbound Usage Goal - Distribute partitions according to inbound/outbound network traffic
  - Topic Replica Distribution - Maintain an even distribution of a topic's partitions across brokers
  - More goals listed at: [Cruise Control GitHub](#)

# Chapter Topics

---

## Managing Kafka

- Managing Kafka Topic Storage
- Optional Instructor-Led Demonstration: Message Retention Period
- Log Cleanup and Collection
- Rebalancing Partitions
- Cruise Control
- **Hands-On Exercise: Installing Cruise Control**
- Hands-On Exercise: Troubleshooting Kafka Topics
- Unclean Leader Election
- Hands-On Exercise: Unclean Leader Election
- Adding and Removing Brokers
- Hands-On Exercise: Adding and Removing Brokers
- Best Practices
- Essential Points

## Hands-On Exercise: Installing Cruise Control

---

- In this exercise, we will install and perform a rebalance using Cruise Control.
- Exercise directory: `exercise-code/installing-cruise-control`

# Chapter Topics

---

## Managing Kafka

- Managing Kafka Topic Storage
- Optional Instructor-Led Demonstration: Message Retention Period
- Log Cleanup and Collection
- Rebalancing Partitions
- Cruise Control
- Hands-On Exercise: Installing Cruise Control
- **Hands-On Exercise: Troubleshooting Kafka Topics**
- Unclean Leader Election
- Hands-On Exercise: Unclean Leader Election
- Adding and Removing Brokers
- Hands-On Exercise: Adding and Removing Brokers
- Best Practices
- 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

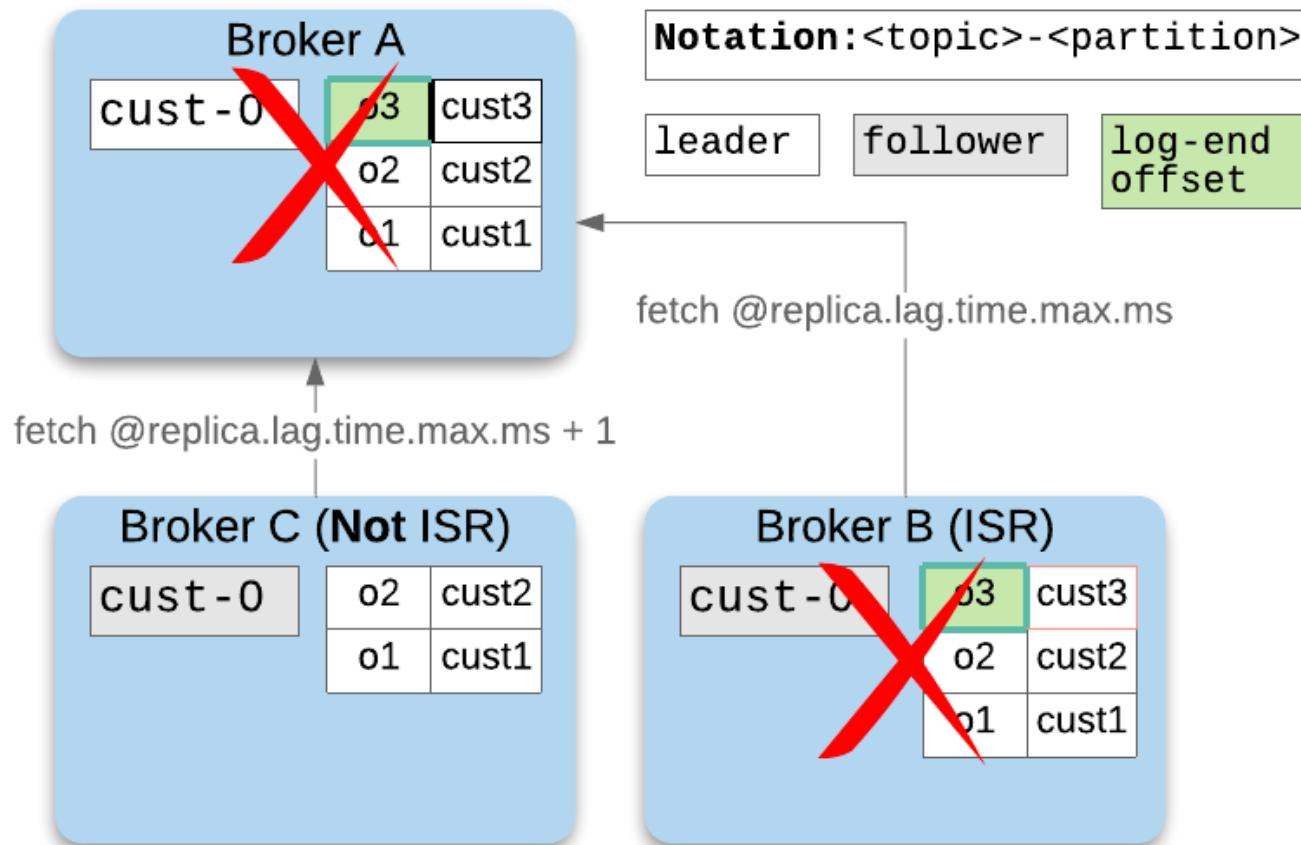
---

## Managing Kafka

- Managing Kafka Topic Storage
- Optional Instructor-Led Demonstration: Message Retention Period
- Log Cleanup and Collection
- Rebalancing Partitions
- Cruise Control
- Hands-On Exercise: Installing Cruise Control
- Hands-On Exercise: Troubleshooting Kafka Topics
- **Unclean Leader Election**
- Hands-On Exercise: Unclean Leader Election
- Adding and Removing Brokers
- Hands-On Exercise: Adding and Removing Brokers
- Best Practices
- Essential Points

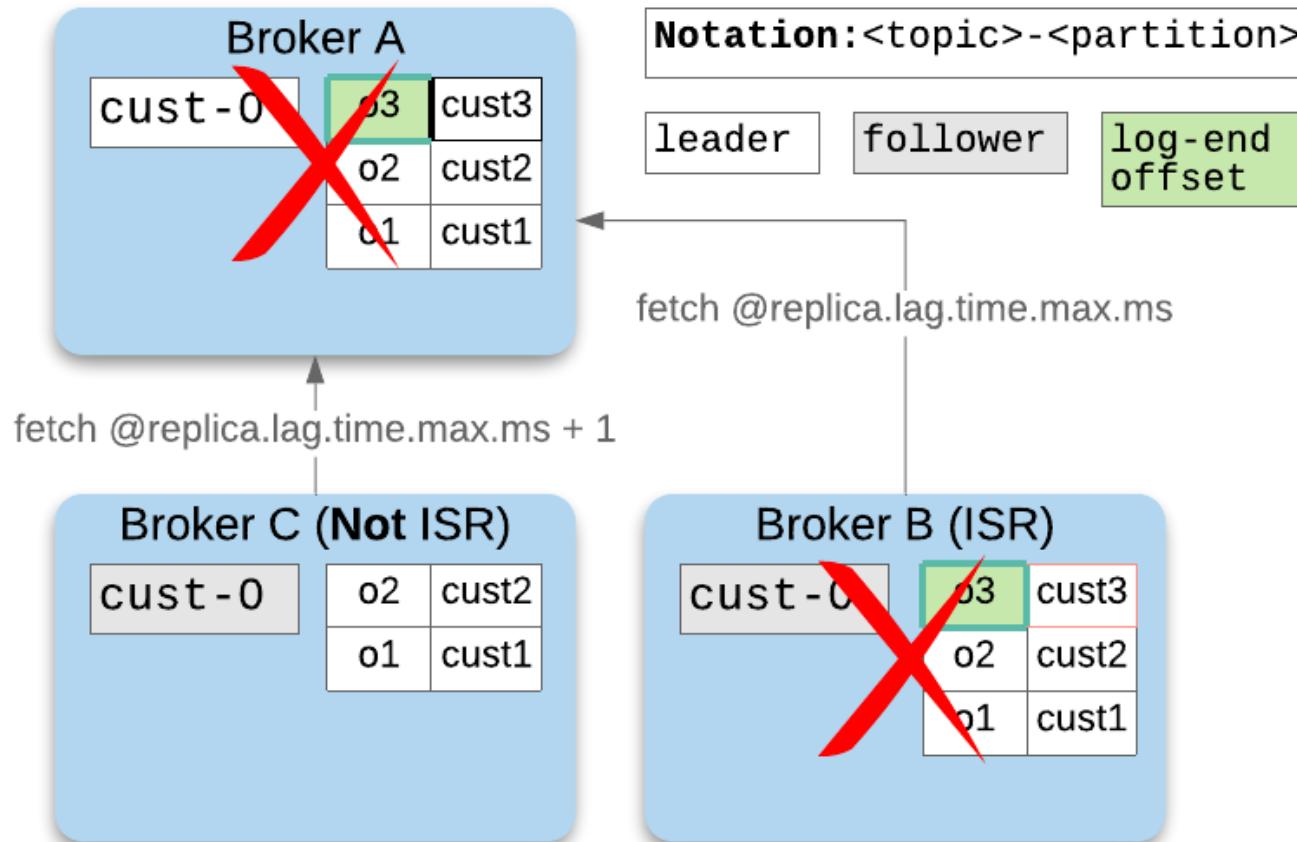
# Unclean Leader Election

- Broker C is the only available broker for a topic+partition
  - However, Broker C is *unclean* due to its being out of sync
  - Kafka will not (by default) automatically elect Broker C as the leader



## Unclean Leader Election, Continued

- Unclean leader election may need to be performed to make a topic available at the expense of data loss



## Unclean Leader Election, Continued

---

- Default value of `unclean.leader.election.enable` is `false`.
  - Topic+partition(s) will simply be offline if leader is dead and followers are out of sync
- An out of sync follower may be assigned as leader manually using `kafka-leader-election` tool.
  - Data loss is probable in this case
- `unclean.leader.election.enable` may be specified per topic
  - Example: Data loss is tolerable and availability is favored for specific topic(s)
- See Exercises -> Kafka Command Summary -> Elect New Leader for Topic / Partitions

# Chapter Topics

---

## Managing Kafka

- Managing Kafka Topic Storage
- Optional Instructor-Led Demonstration: Message Retention Period
- Log Cleanup and Collection
- Rebalancing Partitions
- Cruise Control
- Hands-On Exercise: Installing Cruise Control
- Hands-On Exercise: Troubleshooting Kafka Topics
- Unclean Leader Election
- **Hands-On Exercise: Unclean Leader Election**
- Adding and Removing Brokers
- Hands-On Exercise: Adding and Removing Brokers
- Best Practices
- Essential Points

## Hands-On Exercise: Unclean Leader Election

---

- In this exercise, we will cause a broker to lag and become out of sync.
  - The leader will then experience a failure, and we will allow unclean leader election to make the topic+partition available again (at the expense of data loss)
- Exercise directory: `exercise-code/unclean-leader`

# Chapter Topics

---

## Managing Kafka

- Managing Kafka Topic Storage
- Optional Instructor-Led Demonstration: Message Retention Period
- Log Cleanup and Collection
- Rebalancing Partitions
- Cruise Control
- Hands-On Exercise: Installing Cruise Control
- Hands-On Exercise: Troubleshooting Kafka Topics
- Unclean Leader Election
- Hands-On Exercise: Unclean Leader Election
- **Adding and Removing Brokers**
- Hands-On Exercise: Adding and Removing Brokers
- Best Practices
- Essential Points

# Adding and Removing Brokers: Scenarios

---

- **Brokers are typically added / removed for these reasons:**
  - Increase / decrease in throughput
  - Messages in / out
  - Bytes in / out
  - Increase / decrease in client connections
  - Number of producers / consumers
  - Upgrades
    - Software, Hardware
    - Brokers may be added to increase availability during upgrades

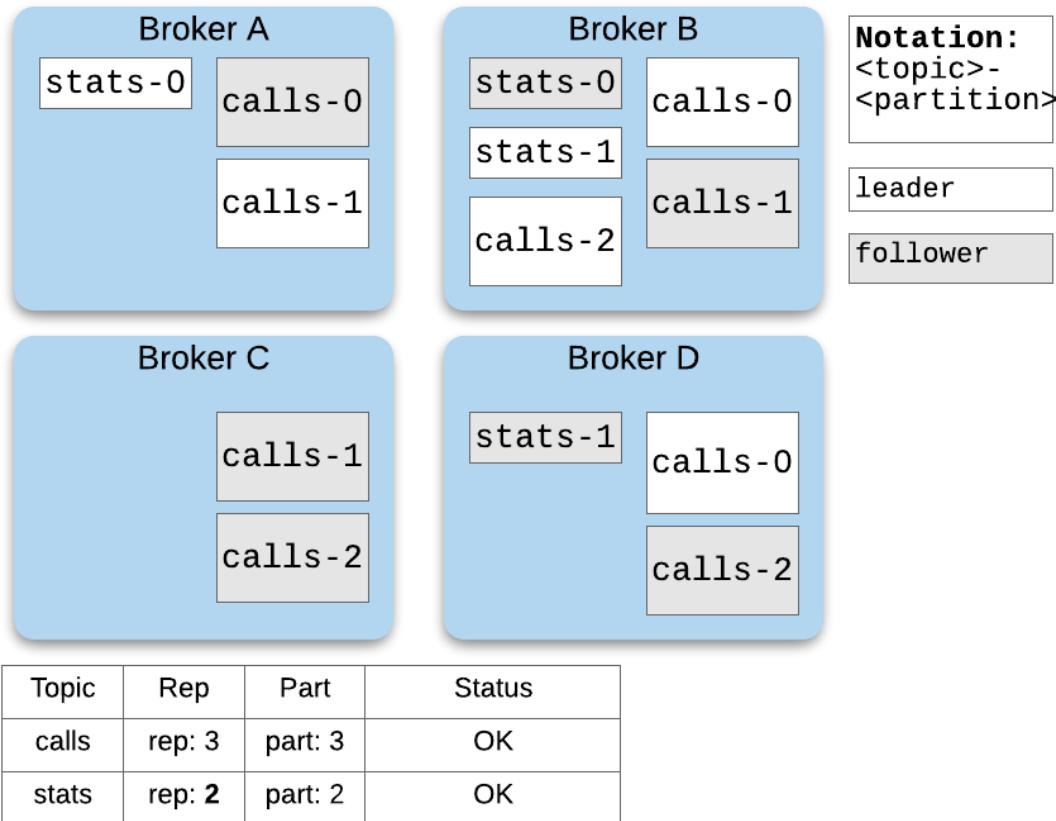
# Adding and Remove Brokers: Considerations

---

- **Adding Brokers:**
  - **Note:** Kafka will not automatically reassign existing partitions to new brokers
- **Removing Brokers / Offline Brokers:**
  - *Leader* assignments will fail over to followers
  - **However:** Kafka will not assign new *followers* in the case of broker failure
- **These behaviors are by design**
  - Avoids possible flooding of bandwidth in the case of *temporary* broker failure
  - Avoids flooding of bandwidth when brokers are added

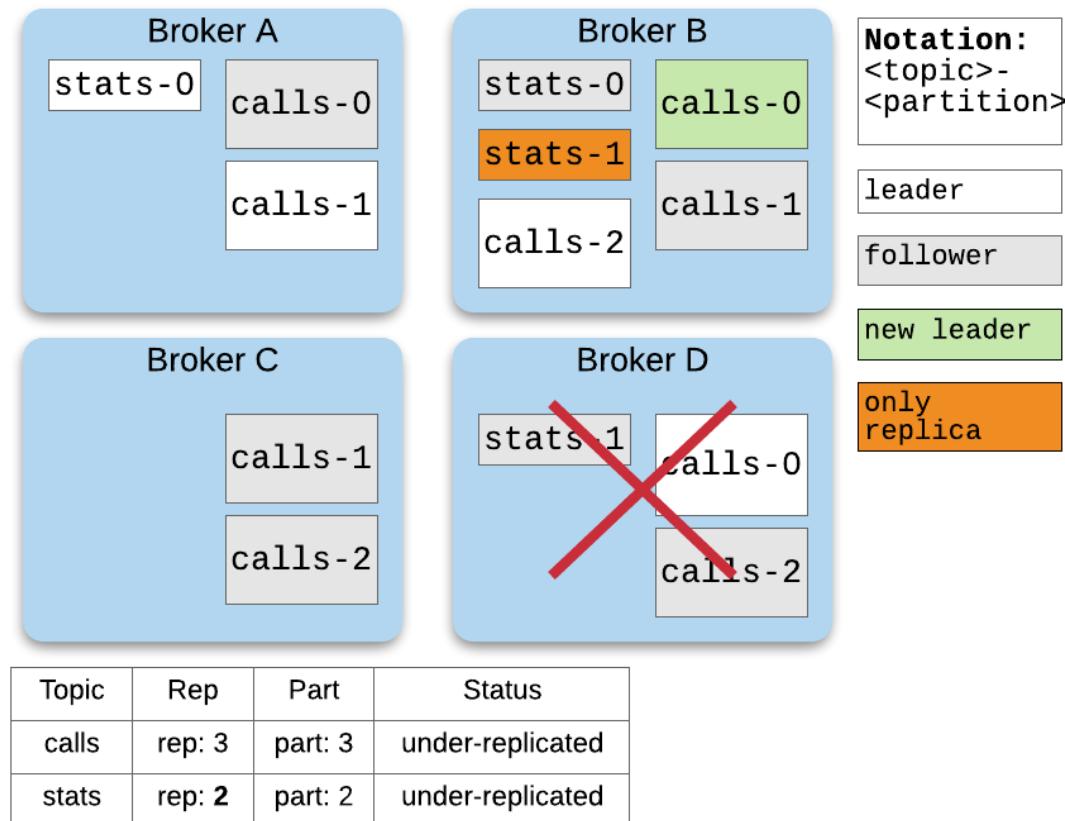
# Failed Brokers or Removing Brokers: Initial State

- What if we just remove Broker D?



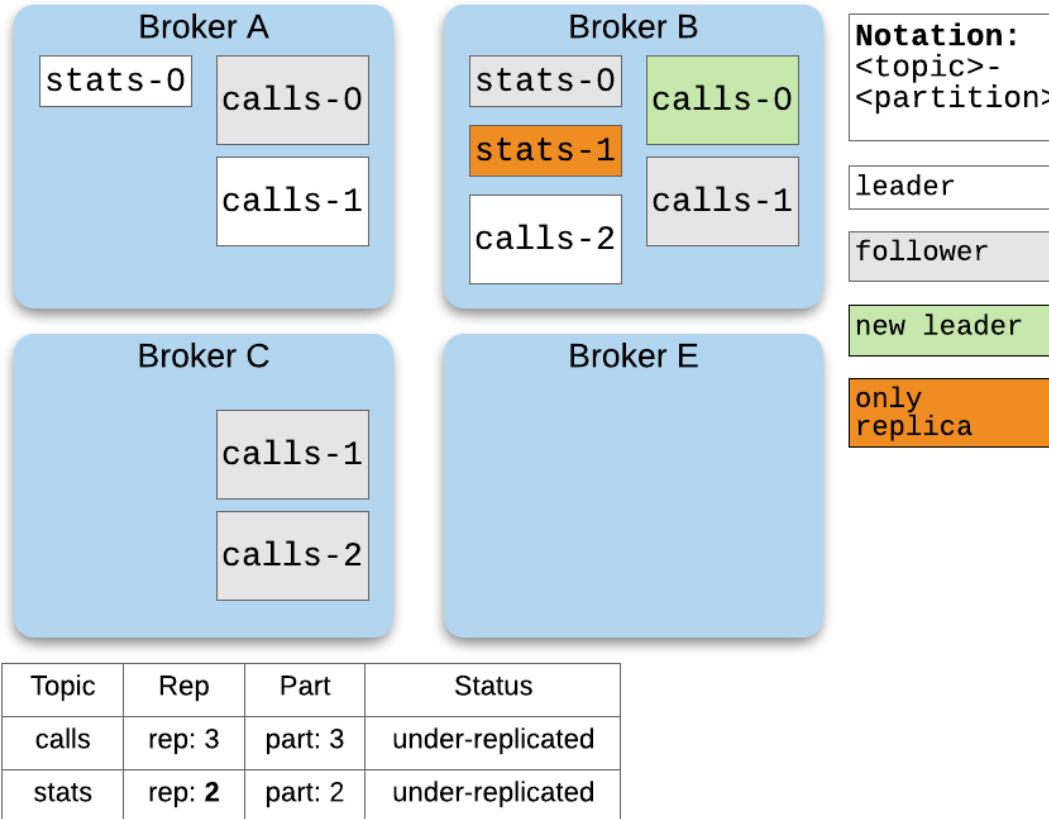
# Failed Brokers / Removing Brokers: Issues

- **Good:** In the example below, leadership for `calls-0` would be moved to broker B
- **Bad:** Under-replicated partitions (followers are not automatically assigned)



# Adding Brokers: Issues

- Adding brokers does not automatically fix under-replicated topics



# Chapter Topics

---

## Managing Kafka

- Managing Kafka Topic Storage
- Optional Instructor-Led Demonstration: Message Retention Period
- Log Cleanup and Collection
- Rebalancing Partitions
- Cruise Control
- Hands-On Exercise: Installing Cruise Control
- Hands-On Exercise: Troubleshooting Kafka Topics
- Unclean Leader Election
- Hands-On Exercise: Unclean Leader Election
- Adding and Removing Brokers
- **Hands-On Exercise: Adding and Removing Brokers**
- Best Practices
- Essential Points

## Hands-On Exercise: Adding and Removing Brokers

---

- In this exercise, you will add and remove brokers.
- Exercise directory: `exercise-code/adding-brokers`

# Chapter Topics

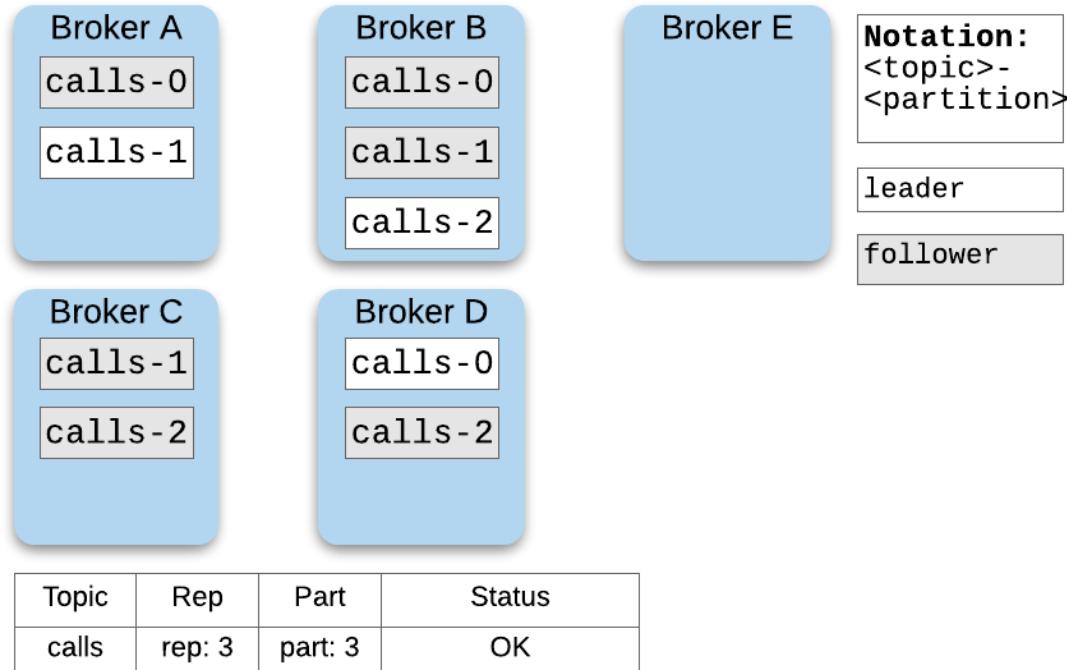
---

## Managing Kafka

- Managing Kafka Topic Storage
- Optional Instructor-Led Demonstration: Message Retention Period
- Log Cleanup and Collection
- Rebalancing Partitions
- Cruise Control
- Hands-On Exercise: Installing Cruise Control
- Hands-On Exercise: Troubleshooting Kafka Topics
- Unclean Leader Election
- Hands-On Exercise: Unclean Leader Election
- Adding and Removing Brokers
- Hands-On Exercise: Adding and Removing Brokers
- **Best Practices**
- Essential Points

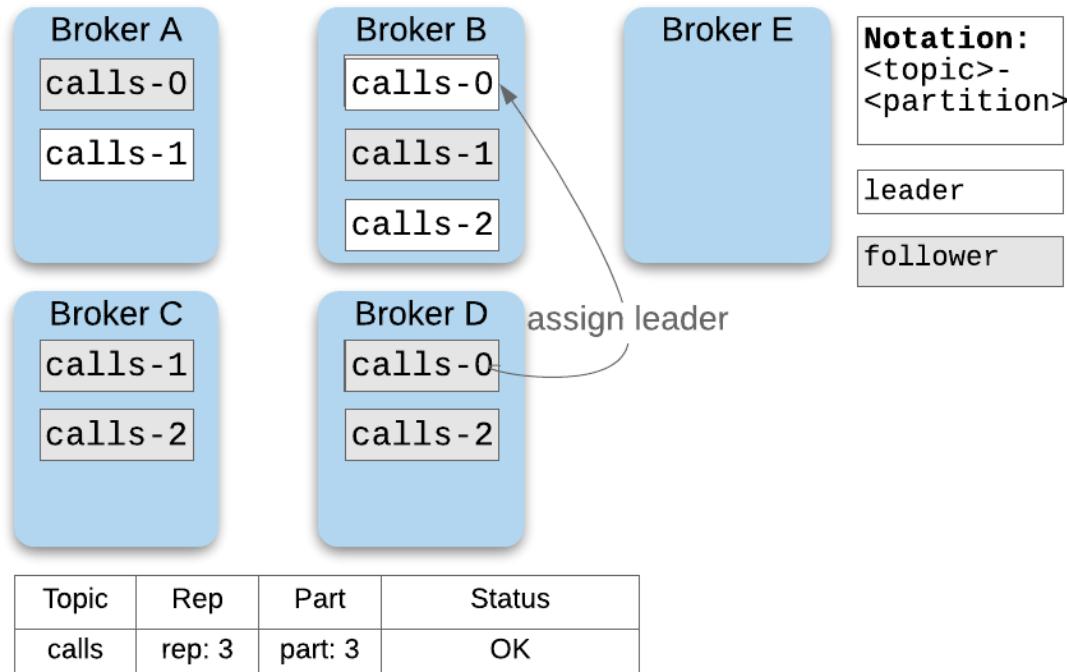
# Best Practice: Rebalance Prior to Removing Broker

- Broker D is to be removed (Optional Broker E is to be added)



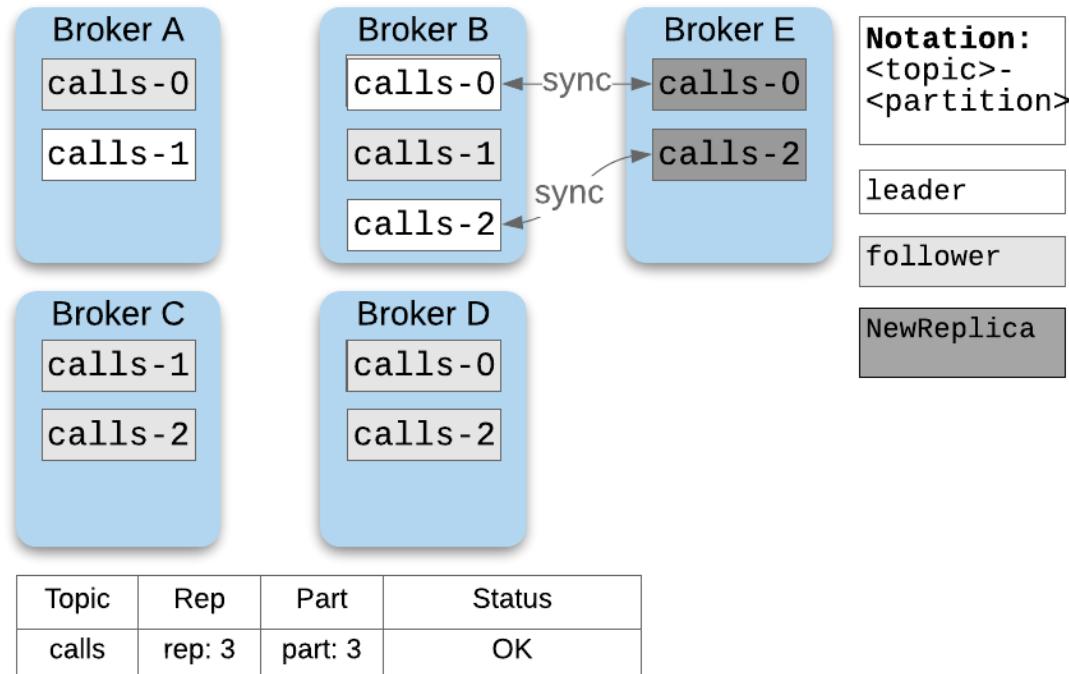
# Rebalance: Assignment of New Leader

- New leader assigned for calls-0. Broker D becomes follower



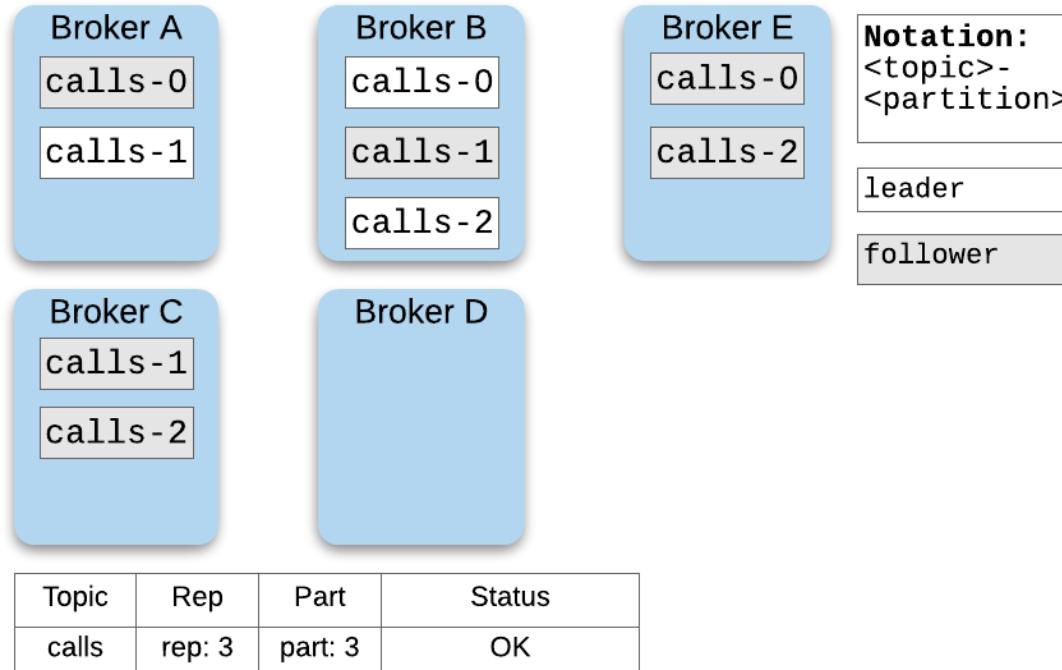
# Rebalance: Assignment of New Replicas

- New replicas assigned for Broker E. Replicas are not available yet ("NewReplica") state



# Rebalance: Complete

- Broker D is ready for Removal. Partitions deleted from Broker D.



# Recap

---

- **Rebalancing**
  - Perform during off-peak times
  - Rebalance Conservatively
    - Calculate data movement (e.g. assignment of new followers)
    - (Leader assignment is "cheap")
  - Suggestion: Rebalance at topic level or even subset of topic+partition
- **Adding / Removing Brokers**
  - Add brokers first
  - "Decommission" brokers using kafka-reassign-partitions (or Cruise Control)
    - Don't just power down old brokers

## Reference

---

- [\*\*Adding and Removing Brokers \(Knowledge Base Article\)\*\*](#)
- [\*\*How to add new Kafka Brokers to a cluster and move existing topics to new Brokers with all topics intact\*\*](#)

# Broker Issues

---

- **Detection: Cloudera Manager**
  - Health tests will typically reveal issues
- **Diagnosis:**
  - Use Events, Health Test Results, Log Files
  - Use Kafka Service -> Brokers -> Broker X -> Charts

# Disk Failures / Migration of Partitions Across Disks at Broker Level

---

- Detection: Cloudera Manager Health Tests
- Migration of Partitions Across Disks
  - JBOD Migration
- Disk Failures on Brokers
  - Handling Disk Failures
- Both of the above tasks are performed using `kafka-reassign-partitions`
- Cruise Control may be another option

# Chapter Topics

---

## Managing Kafka

- Managing Kafka Topic Storage
- Optional Instructor-Led Demonstration: Message Retention Period
- Log Cleanup and Collection
- Rebalancing Partitions
- Cruise Control
- Hands-On Exercise: Installing Cruise Control
- Hands-On Exercise: Troubleshooting Kafka Topics
- Unclean Leader Election
- Hands-On Exercise: Unclean Leader Election
- Adding and Removing Brokers
- Hands-On Exercise: Adding and Removing Brokers
- Best Practices
- **Essential Points**

## Essential Points

---

- Cloudera default settings are a good starting point for most clusters
- Duration and size of Kafka topic storage is influenced by `retention.ms` and `retention.bytes` properties
- Log cleanup can optimize performance by restricting log segments growth size
- Use Cruise Control or the built-in Kafka balancer when Kafka clusters become out of balance with regards to partition assignment
- Unclean leader election may need to be performed to make a topic available at the expense of data loss



# Message Structure, Format, and Versioning

---

Chapter 13

# Course Chapters

---

- Introduction
- Kafka Overview
- Deploying Apache Kafka
- Kafka Command Line Basics
- Using Streams Messaging Manager (SMM)
- Kafka Java API Basics
- Improving Availability through Replication
- Improving Application Scalability
- Improving Application Reliability
- Analyzing Kafka Clusters with SMM
- Monitoring Kafka
- Managing Kafka
- **Message Structure, Format, and Versioning**
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka Cluster
- Conclusion

# Message Structure, Format, and Versioning

---

**By the end of this chapter, you will be able to**

- **Design and use an Avro schema with Kafka**
- **Describe use cases for Schema Registry**

# Chapter Topics

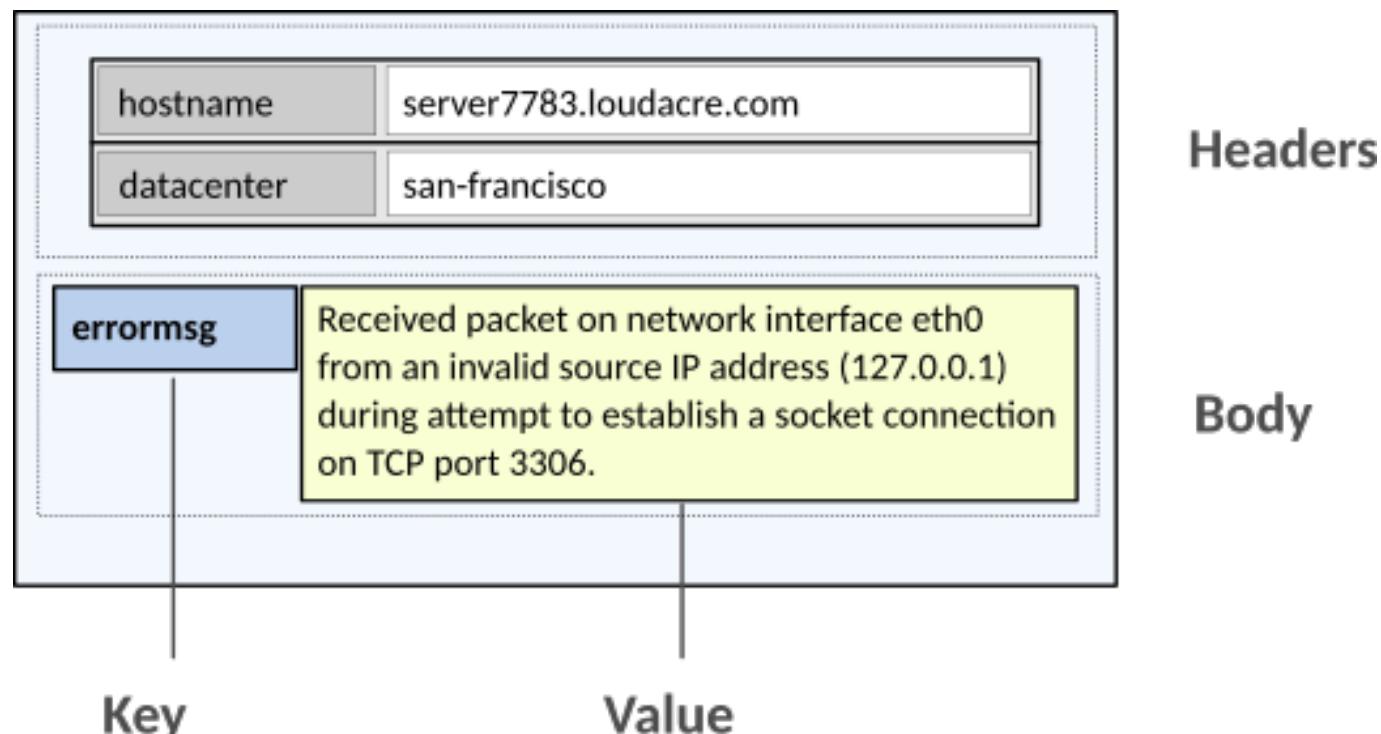
---

## Message Structure, Format, and Versioning

- **Message Structure**
- Schema Registry
- Defining Schemas
- Schema Evolution and Versioning
- Schema Registry Client
- Hands-On Exercise: Using an Avro Schema
- 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

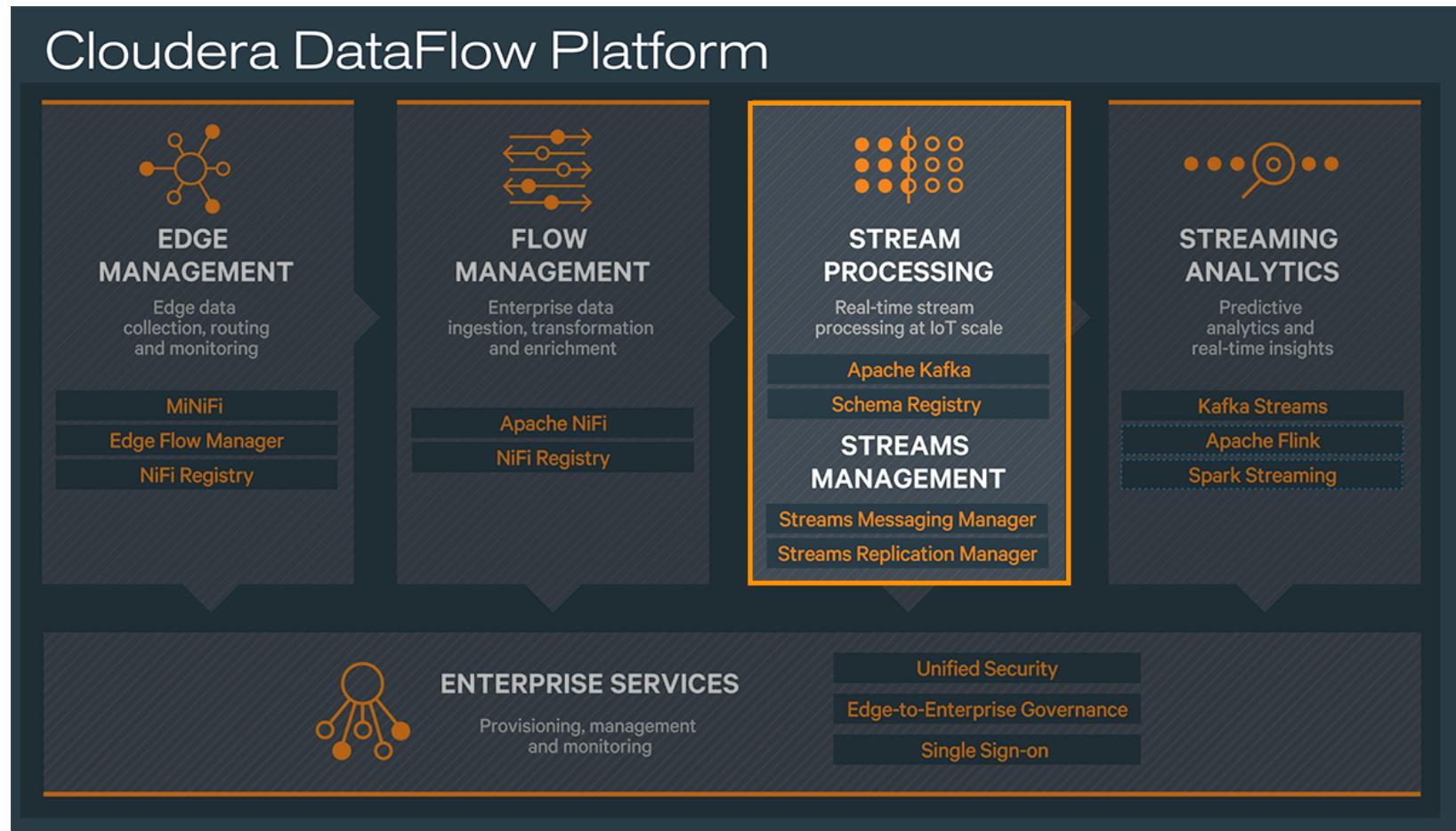
---

## Message Structure, Format, and Versioning

- Message Structure
- **Schema Registry**
- Defining Schemas
- Schema Evolution and Versioning
- Schema Registry Client
- Hands-On Exercise: Using an Avro Schema
- Essential Points

# Cloudera Schema Registry

- Schema Registry is part of the Stream Processing component
  - In the Cloudera DataFlow Platform (CDF)



# Schema Registry Features

- Format validation to enable generic format conversion
- Version management to define relationship between schemas
  - Schemas can be backward and/or forward compatible
  - Consumers and producers can evolve at different rates

The screenshot shows the Schema Registry interface with the title "All Schemas". It features a search bar and a sorting option set to "Last Updated". Three schemas are listed:

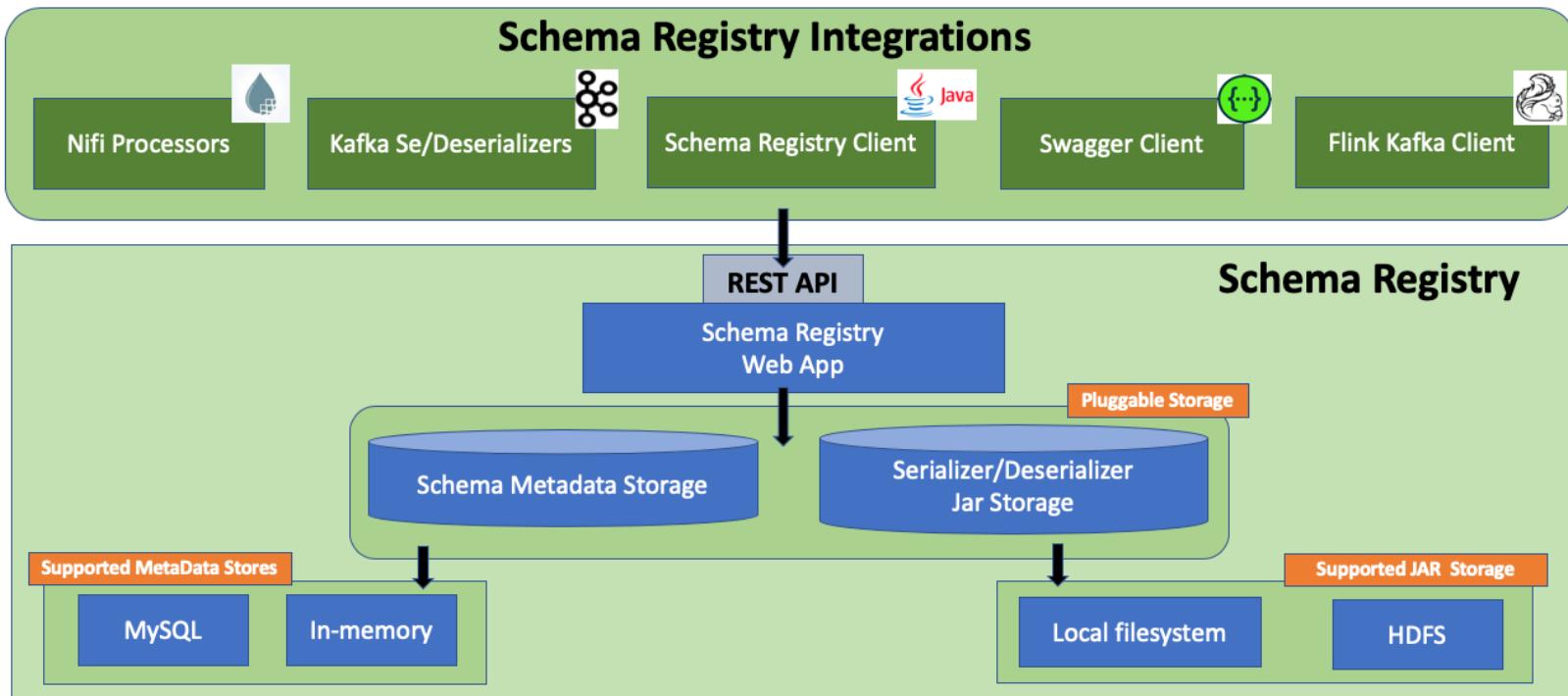
Name	Type	Group	Branch	Serializer & Deserializer
login-failure FORWARD	avro	Kafk...	1 ↗	0
pos-transaction BACKWARD	avro	Kafk...	1 ↙	0
clickstream BACKWARD	avro	Kafk...	1 ↙	0

# Schema Registry Integration

---

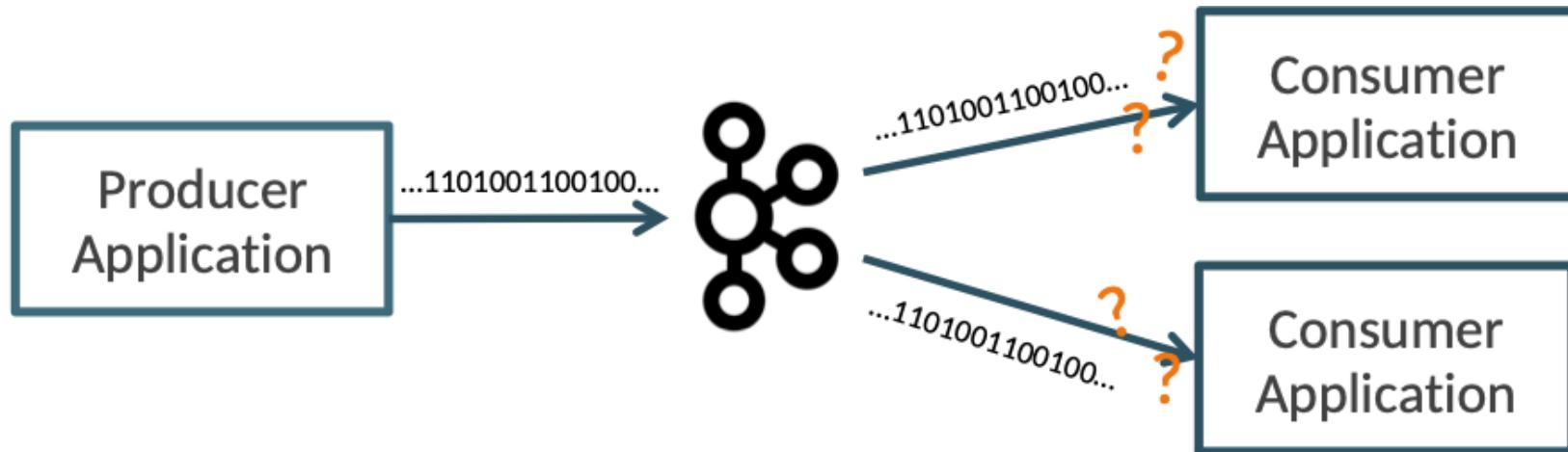
- **Schema Registry is integrated with various Cloudera and Apache tools**
  - Cloudera Flow Management with Apache NiFi
    - Receive and send structured streaming data
    - Perform record-oriented flow processing
  - Apache Kafka
    - Use the Avro/Kafka serializer and deserializer in Kafka client applications
  - Cloudera Streams Messaging Manager
    - View Avro data in Kafka messages
  - Apache Flink
    - Receive, process, and send real time streaming data
- **Integrate with your own applications using**
  - Java API
  - REST API
  - Custom serializers/deserializers

# Schema Registry Component Architecture



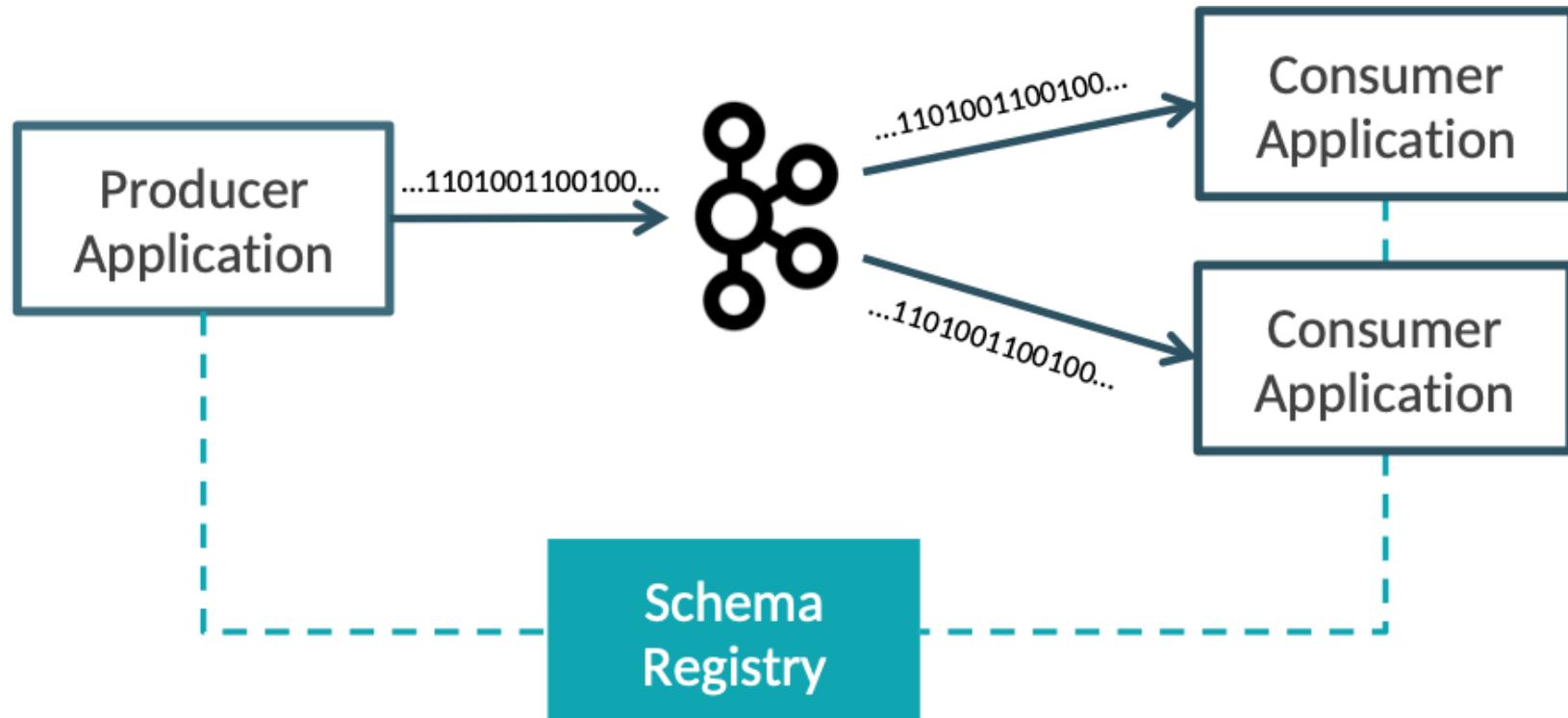
# Kafka Message Data: The Problem

- Kafka takes and publishes bytes
- Applies no structure or data verification



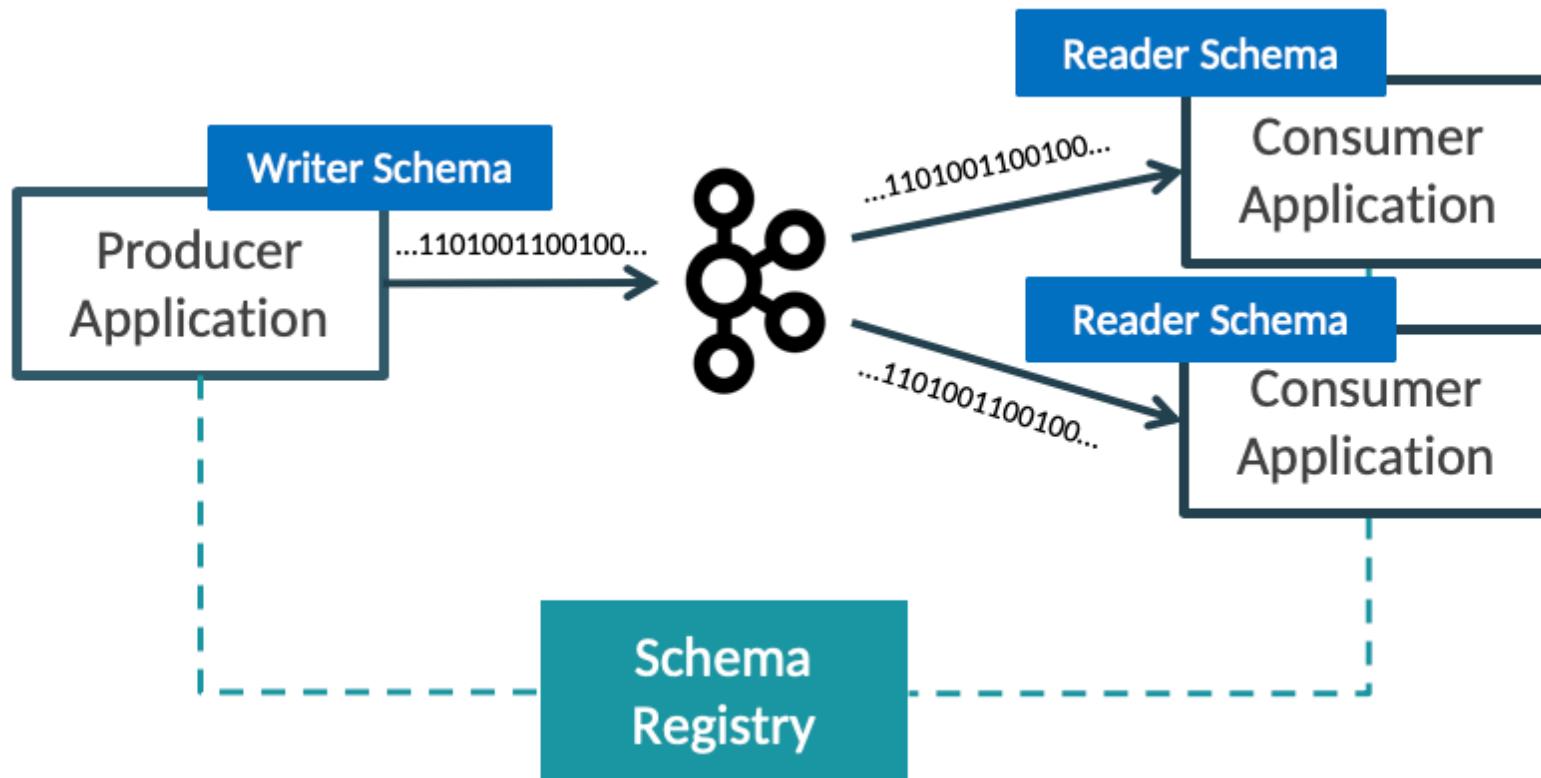
# The Solution: Schema Registry

- Centralized registry for schema
- Provides reusable data definition for producers and consumers



## Writer/Reader Schemas

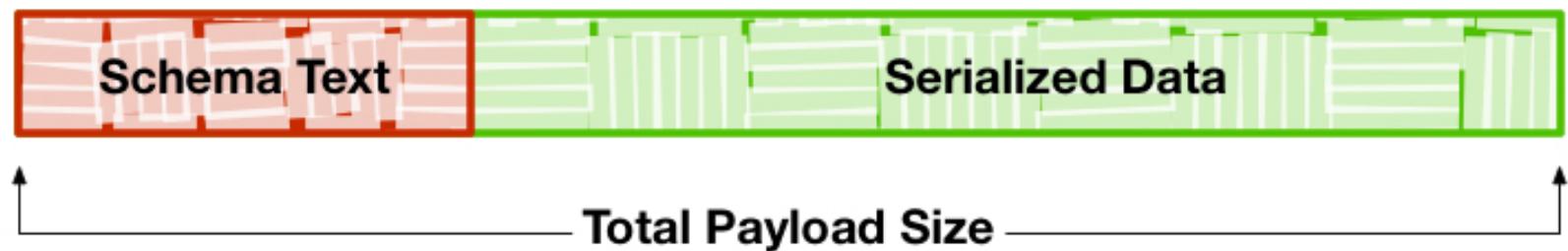
- Senders/Producers use the writer schema while sending payloads using the writer schema
- Receivers use the reader/projection schema to project the received payload written using the writer schema



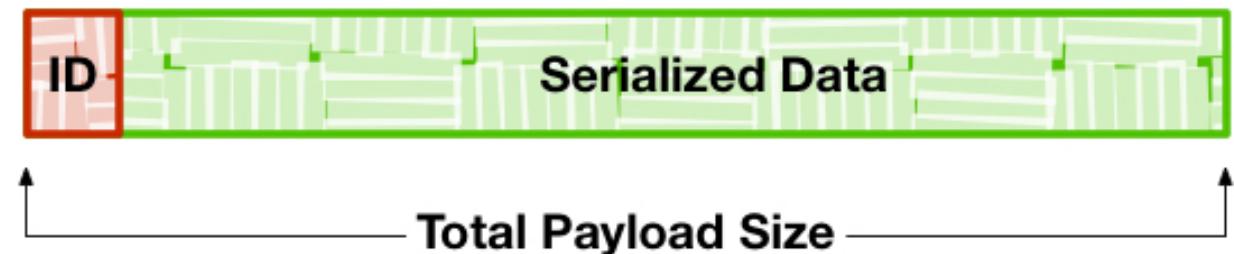
## Separate Schema from Data

---

### Without Schema Registry



### With Schema Registry



# Writer Schemas Embed In Your Data

Time	2019	2020	2021	2022
Avro Schema	quantity	type	date	location
v1	5	blue	May 2019	
v2	20	red	FEB 2020	Seattle
v3	12	purple	APRIL 2021	New York

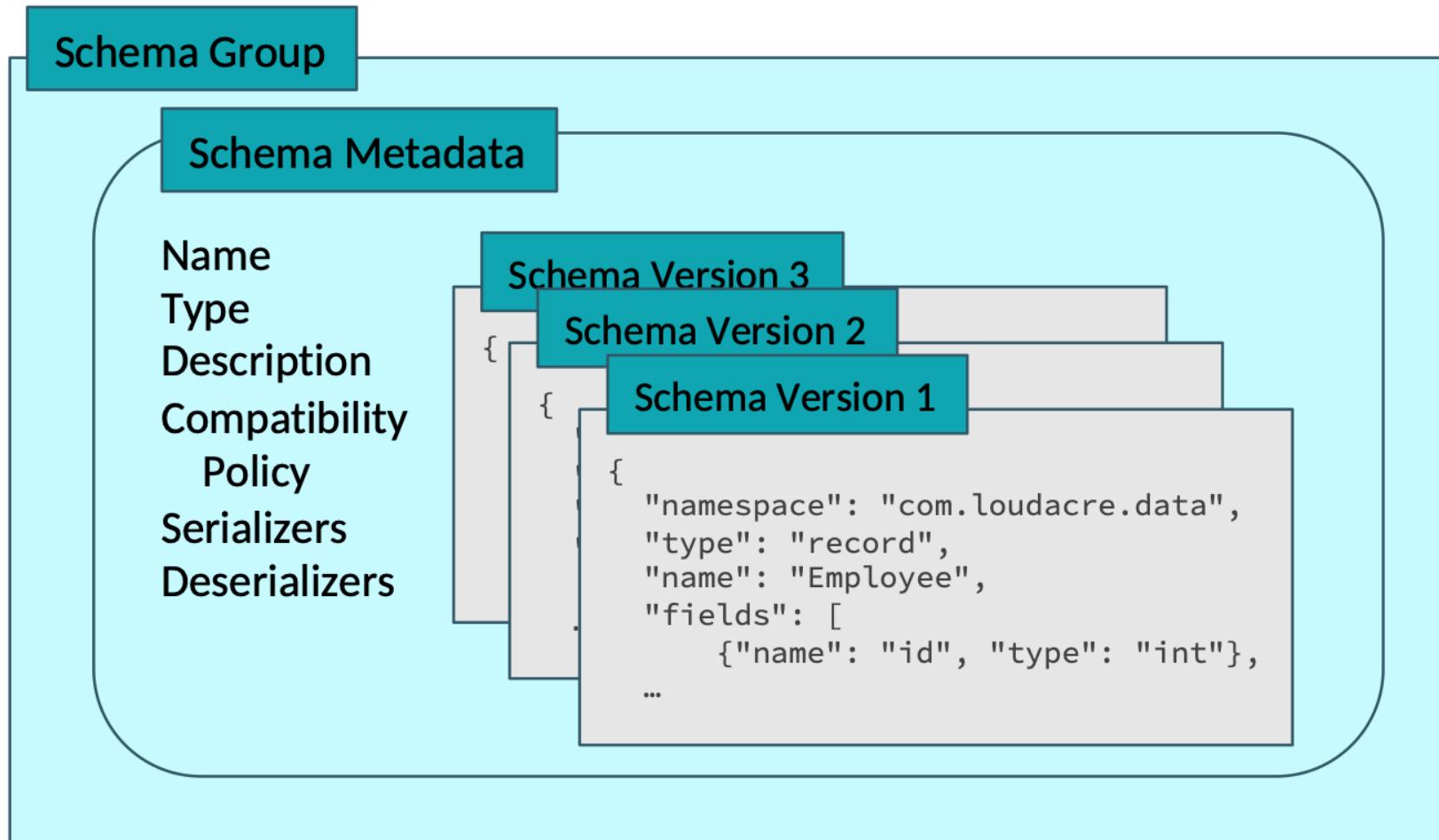
# Chapter Topics

---

## Message Structure, Format, and Versioning

- Message Structure
- Schema Registry
- **Defining Schemas**
- Schema Evolution and Versioning
- Schema Registry Client
- Hands-On Exercise: Using an Avro Schema
- Essential Points

# Schema Definitions in Schema Registry



# Creating a Schema Definition

- Name and description
- Group—helps keep schemas organized
- Type—definition format (only Avro supported at this time)
- Compatibility—forward, backward, none, or all
- Evolve—enable modification of schema

Add New Schema

NAME *	employee
DESCRIPTION *	record in employee DB
TYPE *	Avro schema provider
SCHEMA GROUP *	hr
COMPATIBILITY	BACKWARD
<input checked="" type="checkbox"/> EVOLVE	
SCHEMA TEXT *	
<pre>1 { "namespace": "com.loudacre.data", 2   "type": "record", 3   "name": "Employee", 4   "fields": [ 5     {"name": "id", "type": "int"}, 6     {"name": "name", "type": "string"}, 7     {"name": "title", "type": "string"}, 8     {"name": "bonus", "type": "int"} 9   ] 10 }</pre>	

# Defining a Schema

---

- Schemas are defined using Apache Avro schema specification
- Example: employee record

```
{  
  "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"}  
  ]  
}
```

## Supported Data Types (Simple)

---

- A simple type holds exactly one value
- Avro's string type can be mapped to an Avro utility class, if desired

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 (Complex)

---

- Avro also supports complex types

Name	Description
<b>record</b>	A user-defined type composed of one or more named fields
<b>enum</b>	A specified set of values
<b>array</b>	Zero or more values of the same type
<b>map</b>	Set of key-value pairs; key is <b>string</b> while value is of specified type
<b>union</b>	Exactly one value matching a specified set of types
<b>fixed</b>	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

# Basic Schema Example

---

## ■ Employee record

```
{  
  "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"}  
  ]  
}
```

## Specifying Default Values

---

- Schemas can specify a default value

```
{  
  "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"}  
  ]  
}
```

## Null Values

---

- Null values not allowed unless explicitly specified in the schema
- Example: Allow title and bonus fields to have null values

```
{ "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 is 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)"}
  ]
}
```

# Chapter Topics

---

## Message Structure, Format, and Versioning

- Message Structure
- Schema Registry
- Defining Schemas
- **Schema Evolution and Versioning**
- Schema Registry Client
- Hands-On Exercise: Using an Avro Schema
- Essential Points

# Schema Evolution

---

- **The structure of your data will change over time**
  - Fields can be added, removed, changed, or renamed
- **Schema registry allows you to maintain compatibility between versions**
  - A consumer can use a different schema than the producer

## Schema Evolution: A Practical Example (1)

---

- Imagine that you 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)

---

- You 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)

---

- You could use the new schema to write new data
- But schema is not backward compatible
  - New applications cannot read data that uses old schema
- Make a few schema changes to support compatibility

## Schema Evolution: A Practical Example (4)

---

- Specify an alias for the old name when renaming a field
  - Example: 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 (such as `int` to `long`)
  - Adding aliases for a field

# Schema Evolution: Incompatible Changes

---

- **Some changes 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

# Schema Versioning

- Schemas may be modified
- Schema registry tracks versions
  - Clients can request a specific version of a schema

The screenshot shows the Confluent Schema Registry interface for the 'employee' schema. The top navigation bar indicates the schema name 'employee' and the direction 'BACKWARD'. The schema type is 'avro', group is 'HR', and it is in branch '1' with version 0. The schema code is displayed in a code editor:

```
1 {                                     VERSION 2
2   "namespace": "com.loudacre.da
3   "type": "record",
4   "name": "Employee",
5   "fields": [
6     {
7       "name": "id",
8       "type": "int"
9     },
10    {
11      "name": "name",
12      "type": "string"
13    },
14    {
15      "name": "hire_date",
16      "type": "date"
17    }
18  ]
19}
```

The schema is currently in 'MASTER' branch. To the right, there is a 'CHANGE LOG' section showing two versions: v2 (26s ago, Enabled) and v1 (1h 9m 10s ago, Enabled). A 'COMPARE VERSIONS' button is also present.

# Backward Compatibility

---

- New versions of a schema should be compatible with earlier versions of the schema
- Data written from earlier versions of the schema, can be read with a new version of the schema

v1

```
{  
  "type" : "record",  
  "name" : "book",  
  "namespace": "registry.example",  
  "fields" : [  
    { "name" : "id",  
      "type" : "string"  
    },  
    {  
      "name" : "color"  
      "type" : "string"  
    }  
  ]  
}
```

v2

```
{  
  "type" : "record",  
  "name" : "book",  
  "namespace": "registry.example",  
  "fields" : [  
    { "name" : "id",  
      "type" : "string"  
    },  
    {  
      "name" : "color"  
      "type" : "string"  
    },  
    {  
      "name" : "pages"  
      "type" : "int"  
      "default" : -1  
    }  
  ]  
}
```

# Forward Compatibility

---

- Existing schema is compatible with the future versions of the schema
- Means data written from the new version of the schema can still read with the old version of the schema
- Cannot drop any fields but new fields drop on old schemas for forward compatibility

```
v1
{
  "type" : "record",
  "name" : "book",
  "namespace" : "registry.example",
  "fields" : [
    {
      "name" : "id",
      "type" : "string"
    },
    {
      "name" : "color"
      "type" : "string"
    }
  ]
}

v2
{
  "type" : "record",
  "name" : "book",
  "namespace": "registry.example",
  "fields" : [
    {
      "name" : "id",
      "type" : "string"
    },
    {
      "name" : "color"
      "type" : "string"
    },
    {
      "name" : "pages"
      "type" : "int"
    }
  ]
}
```

# Both/Full Compatibility

---

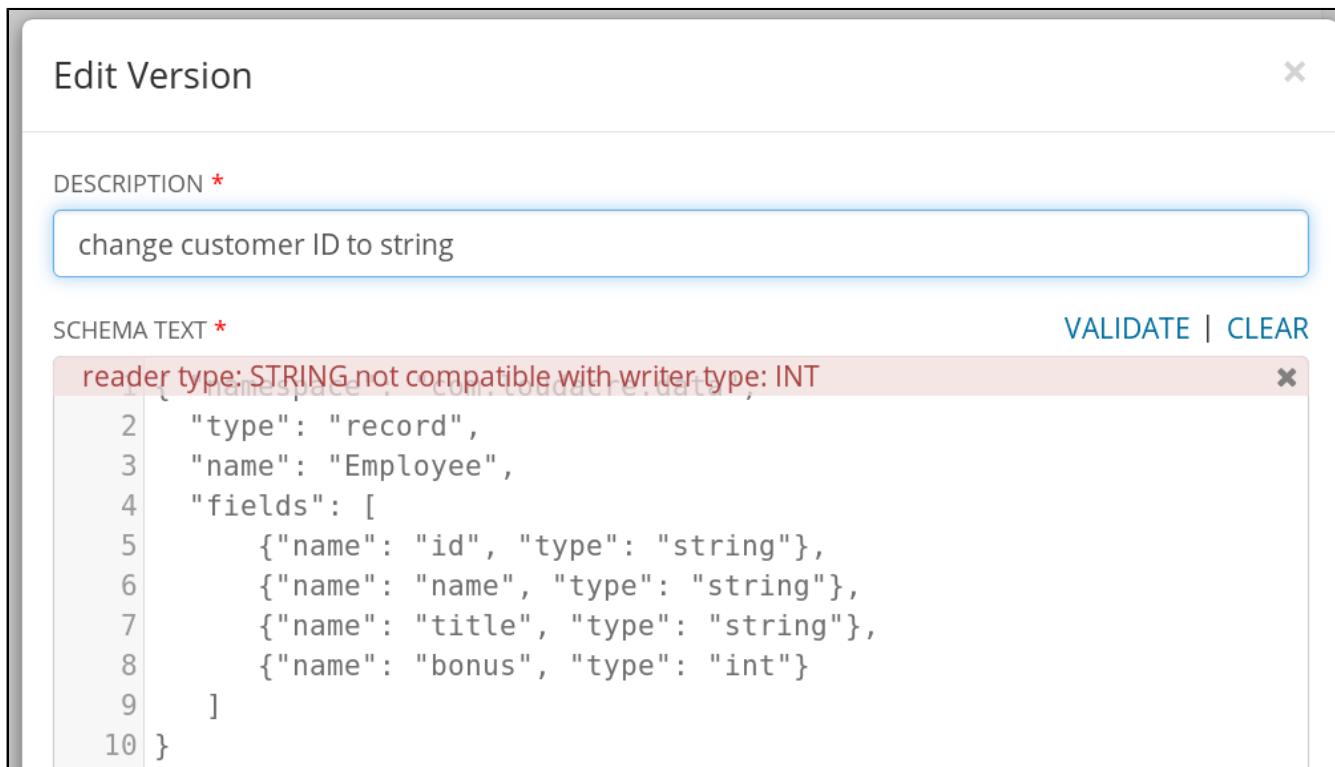
- New versions of the schema provides both backward and forward compatibility
- Schemas can evolve on producer/consumer side
- Declare all fields with default values

```
v1
{
  "type" : "record",
  "name" : "book",
  "namespace" : "registry.example",
  "fields" : [
    {
      "name" : "id",
      "type" : "string"
    },
    {
      "name" : "color"
      "type" : "string"
    }
  ]
}

v2
{
  "type" : "record",
  "name" : "book",
  "namespace": "registry.example",
  "fields" : [
    {
      "name" : "id",
      "type" : "string"
    },
    {
      "name" : "color"
      "type" : "string"
    },
    {
      "name" : "pages"
      "type" : "int"
      "default" : -1
    },
    {
      "name" : "title"
      "type" : "string"
      "default" : ""
    }
  ]
}
```

# Compatibility Validation

- Schema Registry will enforce forward and/or backward compatibility
- Example: Changing a field's data type from int to string is not allowed



# Chapter Topics

---

## Message Structure, Format, and Versioning

- Message Structure
- Schema Registry
- Defining Schemas
- Schema Evolution and Versioning
- **Schema Registry Client**
- Hands-On Exercise: Using an Avro Schema
- Essential Points

# Schema Registry API

---

- **Schema Registry API enables flexibility and custom solutions**
  - Any client via REST or Java API can interact with schema registry
  - Secure interaction for REST and Java API using Kerberos
  - Reduce development time bypassing manual schema management
- **Example custom solutions**
  - Automatic schemas creation for new data patterns
  - Detect new content deltas and trigger generation of new schema versions
  - Register a new serializer/deserializer for new data patterns

# REST API Capabilities

---

- **Schemas**
  - List schemas, download schemas
  - Create, update, fork, merge, archive, and delete schemas
  - Disable and enable schemas or schema versions
  - Create new schema versions
  - Search filters for schemas
- **Serializer/Deserializer**
  - Add serializers, get serializers
  - Bind serializer to schema

# Schema Registry Swagger REST UI

- <https://registryserver:7788/swagger>

The screenshot shows the Swagger UI interface for the Schema Registry API. The title bar includes tabs for 'Schema Registry' and 'Swagger UI'. The main content area is titled '1. Schema'.

**Operations:**

- GET /api/v1/schemaregistry/schemas**: Get list of schemas by filtering with the given query parameters.
- POST /api/v1/schemaregistry/schemas**: Create a schema if it does not already exist.
- GET /api/v1/schemaregistry/schemas/{name}**: Get schema information for the given schema name.

**Parameters:**

Name	Description
name <span style="color:red;">*</span> required string (path)	Schema name activation_event

**Responses:**

Code	Description
200	successful operation

**Example Value | Model**

```
{ "schemaMetadata": { "type": "string", "format": "string", "name": "string", "description": "string" } }
```

# SchemaRegistryClient Java Client Connect

```
public class SampleSchemaRegistryClientApp{
    public static final String DEFAULT_SCHEMA_REG_URL = "http://
localhost:9090/api/v1";

    private final Map <String, Object> config;
    private final SchemaRegistryClient schemaRegistryClient;

    public SampleSchemaRegistryClientApp() {
        config = createConfig(DEFAULT_SCHEMA_REG_URL);
        schemaRegistryClient = new SchemaRegistryClient(config);
        String schemaFileName = "/device.avsc";
        String schema1 = getSchema(schemaFileName);
    }
}
```

# New Schema Code Examples

---

- Registering a new schema

```
SchemaIdVersion v1 =  
    schemaRegistryClient.addSchemaVersion(schemaMetadata,  
        new SchemaVersion(schema1, "Initial version of  
            the schema"));
```

- Adding a new schema

```
String schema2 = getSchema("/device-next.avsc");  
  
SchemaVersion schemaInfo2 =  
    new SchemaVersion(schema2, "second version");  
  
SchemaIdVersion v2 =  
    schemaRegistryClient.addSchemaVersion(schemaMetadata,  
        schemaInfo2);
```

## Schema Version Code Examples

---

- Get a specific schema version

```
String schemaName = schemaMetadata.getName();
SchemaVersionInfo svInf =
    schemaRegistryClient.getSchemaVersionInfo(
        new SchemaVersionKey(schemaName, v2.getVersion()));
```

- Get the latest schema version

```
SchemaVersionInfo latest =
    schemaRegistryClient.getLatestSchemaVersionInfo(schemaName);
```

# Chapter Topics

---

## Message Structure, Format, and Versioning

- Message Structure
- Schema Registry
- Defining Schemas
- Schema Evolution and Versioning
- Schema Registry Client
- **Hands-On Exercise: Using an Avro Schema**
- Essential Points

## Hands-On Exercise: Using an Avro Schema

---

- 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
- Schema Registry
- Defining Schemas
- Schema Evolution and Versioning
- Schema Registry Client
- Hands-On Exercise: Using an Avro Schema
- 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
- **Cloudera Schema Registry**
  - Provides a central location for managing schemas
  - Offers a UI for defining and modifying schemas
- **Schemas are defined using Apache Avro schema specification**
  - Schemas can be modified (evolved) so that they are backward and/or forward compatible

# Bibliography

---

The following offer more information on topics discussed in this chapter

- [Javadoc for Kafka serialization Package](#)
- [Apache Avro: Getting Started Guide \(Java\)](#)
- [Integrating Kafka and Schema Registry](#)
- [Adding a New Schema](#)



# Improving Application Performance

---

Chapter 14

# Course Chapters

---

- Introduction
- Kafka Overview
- Deploying Apache Kafka
- Kafka Command Line Basics
- Using Streams Messaging Manager (SMM)
- Kafka Java API Basics
- Improving Availability through Replication
- Improving Application Scalability
- Improving Application Reliability
- Analyzing Kafka Clusters with SMM
- Monitoring Kafka
- Managing Kafka
- Message Structure, Format, and Versioning
- **Improving Application Performance**
- Improving Kafka Service Performance
- Securing the Kafka Cluster
- Conclusion

# Application Performance

---

**By the end of this chapter, you will be able to improve application performance using**

- **Message size**
- **Batching**
- **Compression**

# Chapter Topics

---

## Improving Application Performance

- **Message Size**
- Batching
- Compression
- Hands-On Exercise: Observing How Compression Affects Performance
- Essential Points

# Message Size

---

- Please see the following links:
  - [Handling Large Messages](#)
  - Latency versus throughput
  - Batch sizes
  - Linger ms (batch waiting period)
  - [Apache Kafka Overview](#) Section: What's a good size of a Kafka record if I care about performance and stability?

# Chapter Topics

---

## Improving Application Performance

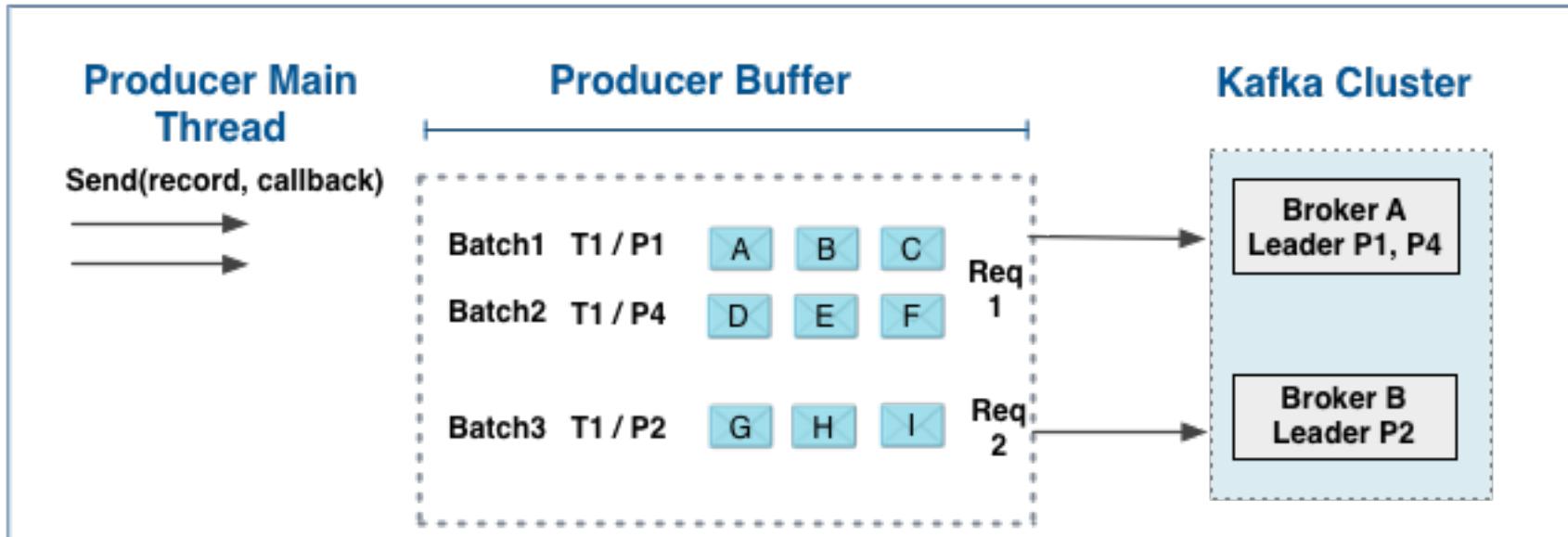
- Message Size
- **Batching**
- Compression
- Hands-On Exercise: Observing How Compression Affects Performance
- Essential Points

## 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
<code>batch.size</code>	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
<code>linger.ms</code>	Amount of time to wait prior to sending a batch of records	0	Adjust based on desired latency
<code>max.request.size</code>	Maximum size of a request in bytes sent to a broker	1048576	
<code>buffer.memory</code>	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
<code>max.block.ms</code>	Maximum number of milliseconds for <code>send()</code> threads to block if send buffer is full	60_000	<code>Long.MAX_VALUE</code> (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
- What's a good size of a Kafka record

# Chapter Topics

---

## Improving Application Performance

- Message Size
- Batching
- **Compression**
- Hands-On Exercise: Observing How Compression Affects Performance
- Essential Points

# Compression

---

- Compression algorithms may be applied to Kafka messages
- Typically, compression is applied to batches of messages
- Potential benefits:
  - Less network bandwidth required to send and receive messages
  - Less storage space required on brokers
- Tradeoffs of compression
  - CPU cycles are used to apply compression and decompression of messages
  - Higher latency

## Messages:



~ 16 KB Raw Data

A,B,C,D

~ 4 KB Compressed

# End to End Compression

- Producers compress messages prior to sending
- Consumers decompress messages when processing

Topic: Orders  
Partition: 0

## Producer A (No Compression)

Batch 1 (16 KB) A B C

Here's 3 messages

## Producer B (Using Compression)

Batch 1 (16 KB) D E F

Compress messages

4 KB GZIP (D,E,F)

## Kafka Cluster

Broker

Disk

A B C

4 KB GZIP (D,E,F)

## Consumer

A B C

4 KB GZIP (D,E,F)

DEcompress  
messages

D E F

## Enabling End to End Compression

---

- In producer code, set `compression.type` to a supported codec
- Supported Codecs: LZ4, GZip, Snappy, Zstandard

## Additional Option: Compression by Brokers

---

- A compression codec may be specified at the *topic* level
- *Brokers apply compression codec after receiving messages*
- When creating (or altering) a topic, specify `compression.type`
- Supported Codecs: LZ4, GZip, Snappy, Zstandard, producer (default)
  - producer means "retain original compression codec set by producer"

# Topic Level Compression and End to End Compression

---

- You can use both
- End to End compression:
  - Bandwidth is saved from producers to brokers
  - CPU is used by producers to compress messages
- Topic level compression:
  - If producers are not performing compression, no bandwidth is saved from producer to broker
  - If producers are not performing compression, brokers will apply compression
  - May be used as a "fail safe" to ensure that messages are stored using compression

# Testing Compression Performance

---

- **Benchmark both producers *and* consumers**
  - Use "real world" data, producers and consumers if possible
  - Use realistic throughput (e.g. "X messages from producers per second")
  - Benchmark all available codecs
- **Use producer settings `batch.size` and `linger.ms` to tune producers**
- **Benchmark Metrics:**
  - Message latency - how long from producer to consumer
  - Message throughput - messages sent/received per second
  - Storage space required
  - Bandwidth required

# Compression: Resources

---

- [CDP Documentation: Kafka Producers](#)
- [Apache documentation](#)
  - See the End to End Batch Compression section
  - See the Topic Level Compression section

# Chapter Topics

---

## Improving Application Performance

- Message Size
- Batching
- Compression
- **Hands-On Exercise: Observing How Compression Affects Performance**
- 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
- Batching
- Compression
- Hands-On Exercise: Observing How Compression Affects Performance
- **Essential Points**

## Essential Points

---

- Batch records for higher throughput
- Compression results in less network bandwidth to send and receive messages
- Two types of compression
  - End to end
  - Topic level
- Four compression Codecs: none, GZip, LZ4, and Snappy



# Improving Kafka Service Performance

---

Chapter 15

# Course Chapters

---

- Introduction
- Kafka Overview
- Deploying Apache Kafka
- Kafka Command Line Basics
- Using Streams Messaging Manager (SMM)
- Kafka Java API Basics
- Improving Availability through Replication
- Improving Application Scalability
- Improving Application Reliability
- Analyzing Kafka Clusters with SMM
- Monitoring Kafka
- Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance**
- Securing the Kafka Cluster
- Conclusion

# Improving Kafka Service Performance

---

By the end of this chapter, you will be able to

- **Describe the factors to consider when sizing your Kafka cluster**
- **Describe the relationship of the resources required for successful operation of a Kafka cluster**
- **Explain general guidelines for the optimal number of partitions per broker**
- **Learn capacity requirements needed for certain workloads**
- **Identify tunable configuration settings**

# Chapter Topics

---

## Improving Kafka Service Performance

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

# Handling Cluster Growth and Tuning: Resources

---

- See [Reference Architectures](#) corresponding to your hardware/cloud platform
  - See "Networking Parameters" section
- Examples from [CDP Kafka Performance Tuning](#) include the following:
  - [ISR Management](#)
  - [JVM and Garbage Collection](#)

# Chapter Topics

---

## Improving Kafka Service Performance

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

# Cluster Sizing

---

- Kafka requires a detailed analysis of the ingest and egress of data
- Cluster sizing is typically performed by identifying the resource which poses the "bottleneck" and then size according to the resource
  - CPU
  - Network
  - Memory
  - Disk I/O
  - Disk Storage

# Chapter Topics

---

## Improving Kafka Service Performance

- Performance Tuning Strategies for the Administrator
- Cluster Sizing
- **Hands-On Exercise: Planning Capacity Needed for a Use Case**
- 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
- Exercise directory: `exercise-code/capacity-planning`

# Chapter Topics

---

## Improving Kafka Service Performance

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

# Essential Points

---

- **Resource analysis allows you to effectively create a sizing estimate for the Kafka cluster**
  - CPU
  - Memory
  - Number of disks
  - Disk capacity
  - Disk throughput per node
  - Network throughput per node
- **Sizing estimate allows you to specify**
  - Broker hardware configuration
  - Number of brokers
  - Number of partitions per broker



# Securing the Kafka Cluster

---

Chapter 16

# Course Chapters

---

- Introduction
- Kafka Overview
- Deploying Apache Kafka
- Kafka Command Line Basics
- Using Streams Messaging Manager (SMM)
- Kafka Java API Basics
- Improving Availability through Replication
- Improving Application Scalability
- Improving Application Reliability
- Analyzing Kafka Clusters with SMM
- Monitoring Kafka
- Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka Cluster**
- Conclusion

# Securing the Kafka Cluster

---

By the end of this chapter, you will be able to

- **Describe the key features and benefits of encryption**
- **Define authentication principles using Kerberos**
- **Understand Apache Ranger's authorization and audit roles**

# 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

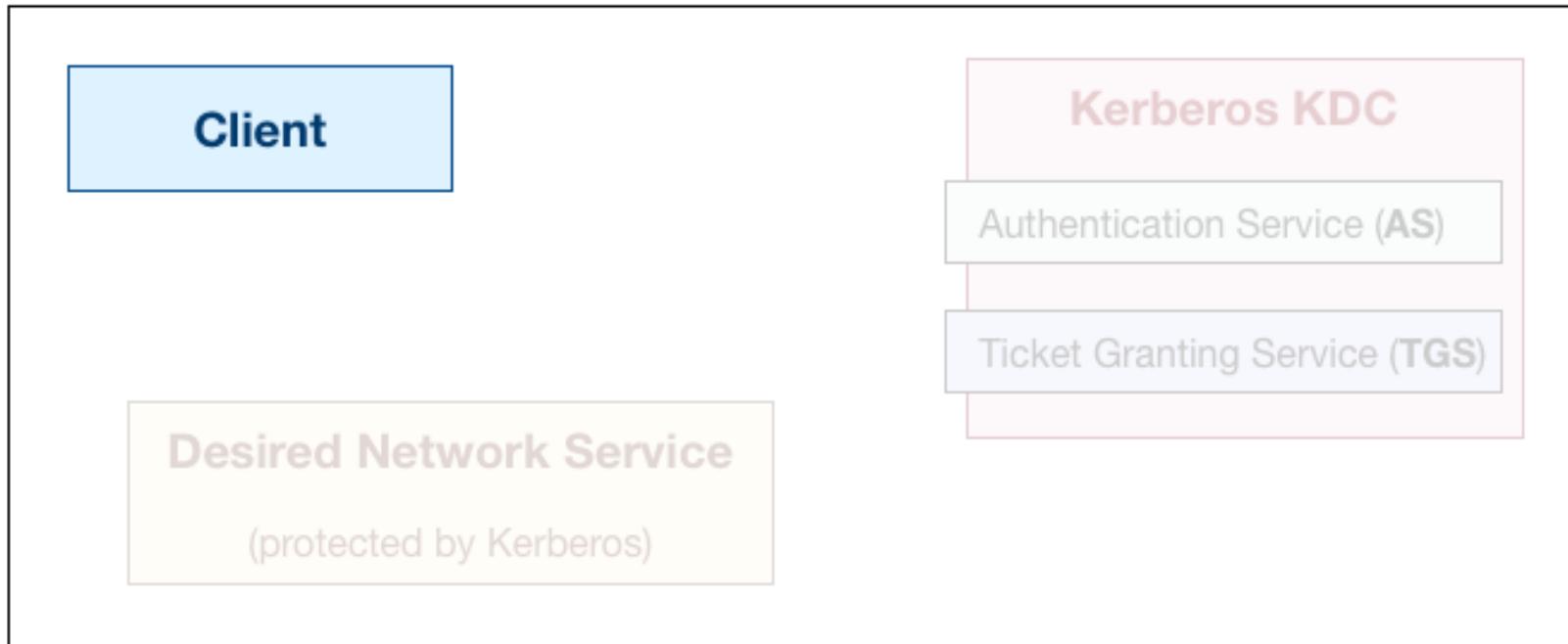
- Encryption
- **Authentication**
- Authorization
- Auditing
- Essential Points

# What Is Kerberos?

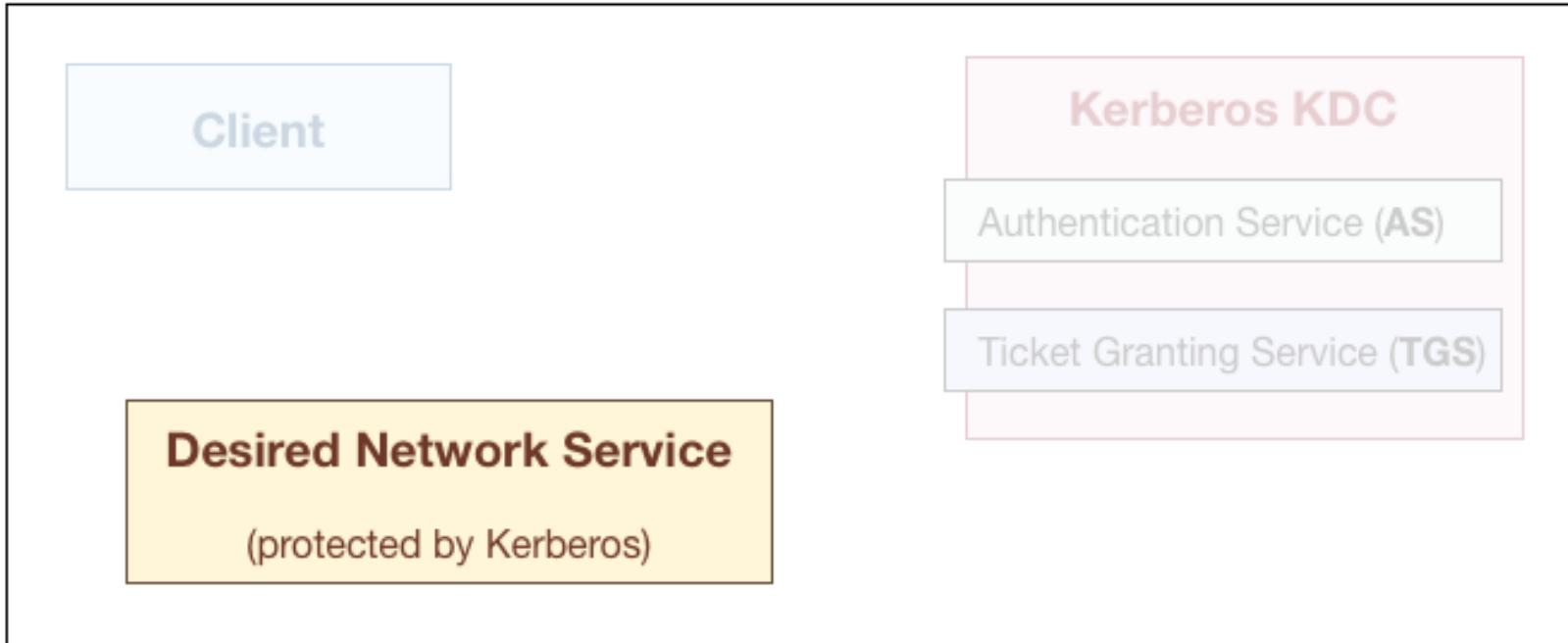
---

- **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 CDP

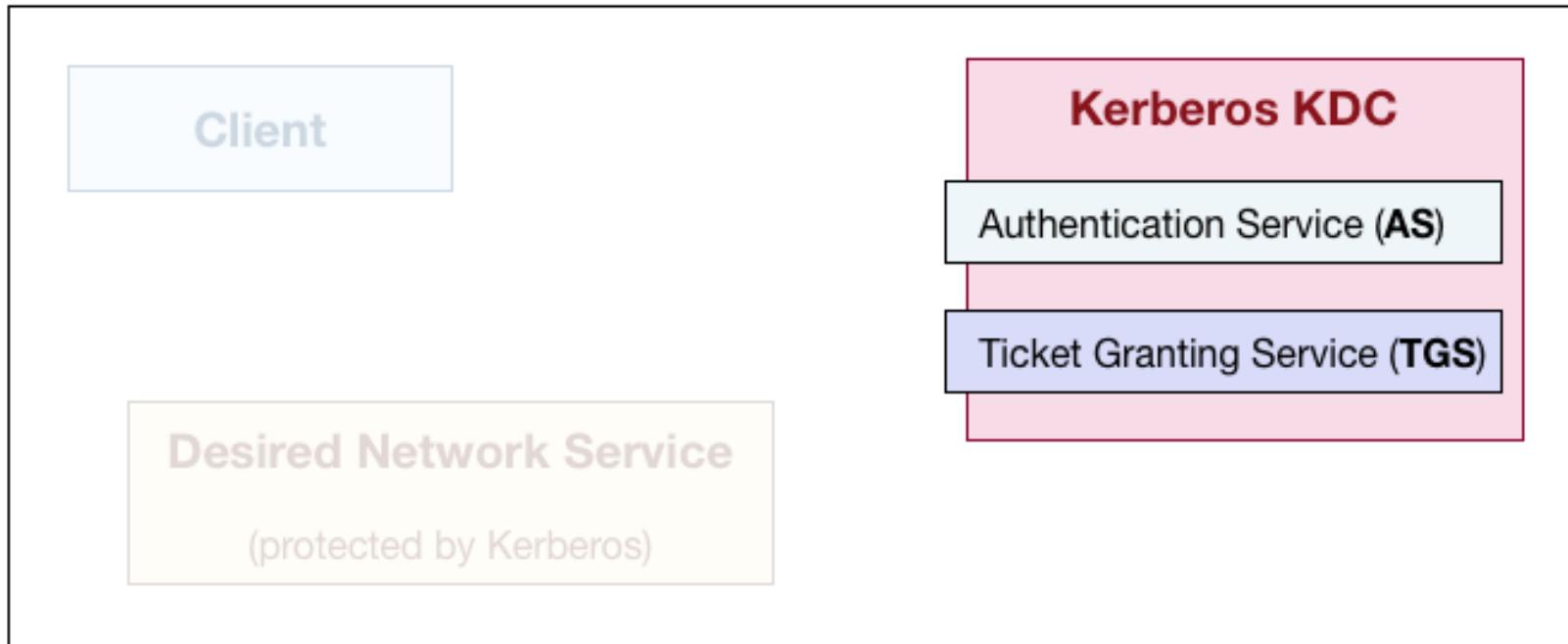
# Kerberos Exchange Participants: Client



# 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
01/10/2021 17:18:15  01/10/2021 03:18:15  krbtgt/kdc.example.com@EXAMPLE.COM
                    renew until 01/17/2021 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,worker3.example.com:9093  
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"`
- In 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

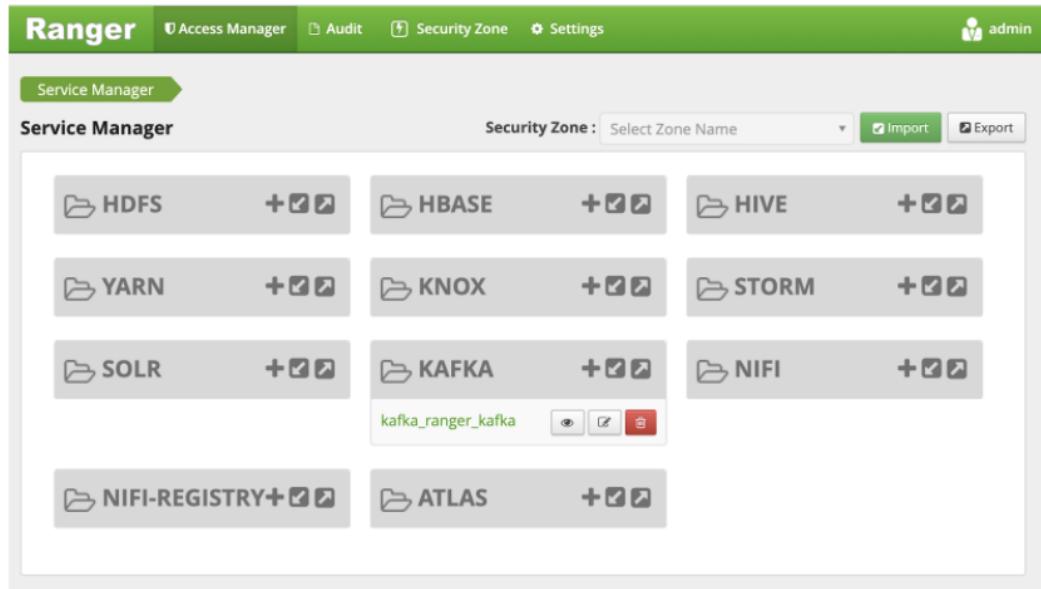
---

- 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

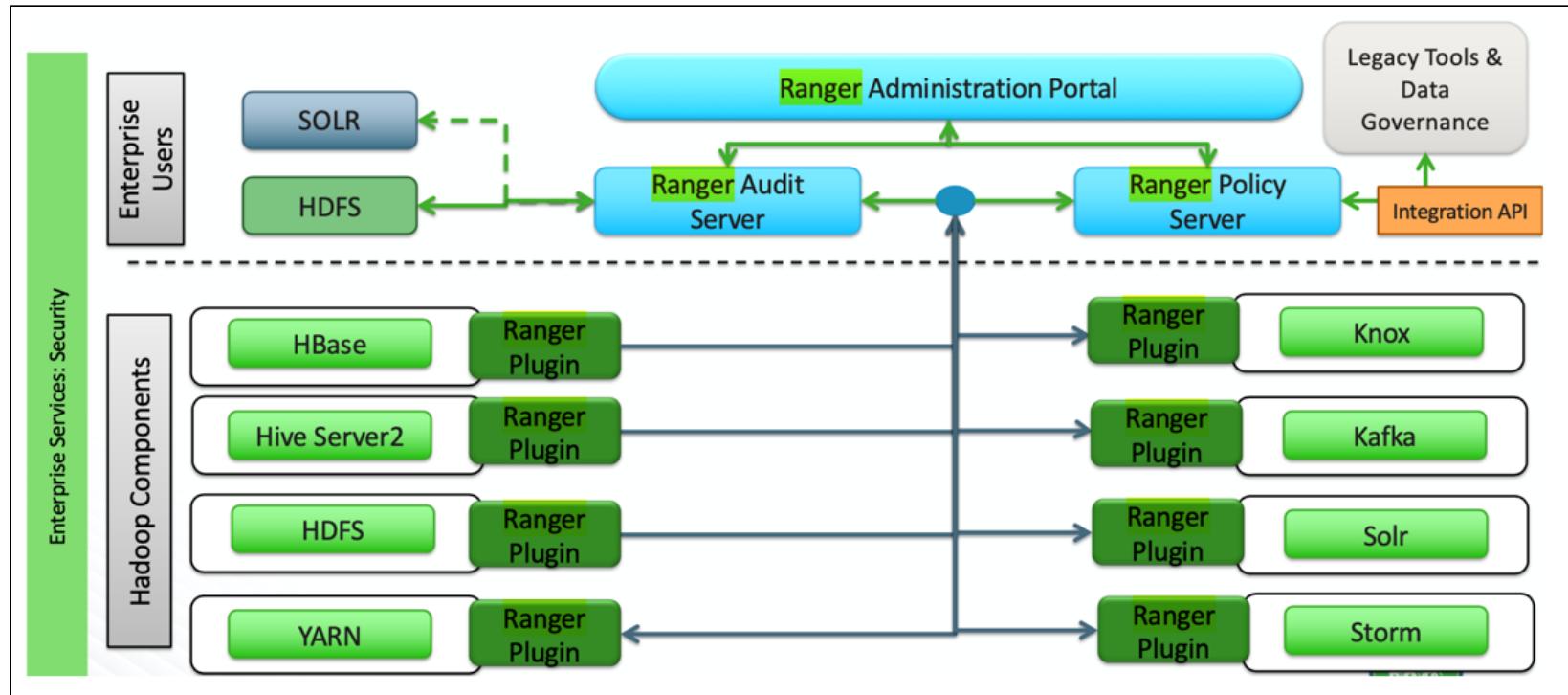
# Apache Ranger?

---

- The comprehensive policy management system across CDP and CDF
- Managed via
  - Browser-based UI
  - REST API
- Requires Kerberos

A screenshot of the Apache Ranger Service Manager interface. The top navigation bar includes tabs for Access Manager, Audit, Security Zone, and Settings, along with a user icon for 'admin'. Below the navigation is a 'Service Manager' section with a 'Security Zone' dropdown set to 'Select Zone Name' and buttons for Import and Export. The main area displays a grid of service icons: HDFS, HBASE, HIVE, YARN, KNOX, STORM, SOLR, KAFKA, NIFI, NIFI-REGISTRY, and ATLAS. Each service has a plus sign and three circular icons next to it. A specific entry for 'kafka\_ranger\_kafka' is highlighted in green at the bottom of the KAFKA row, with three small icons below it: a magnifying glass, a checkmark, and a trash can.

# Ranger Architecture



# Chapter Topics

---

## Securing the Kafka Cluster

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

# Auditing Via Apache Ranger

---

- **Ranger's plug-in architecture supports auditing functionality in addition to policy enforcement**
- **Audit information from all applications is**
  - Co-located together
  - Normalized into the same record format
  - Stored in multiple locations
    - HDFS provides long-term storage of flat files
    - Apache Solr enables high speed searching from the Audit UI

# Ranger Audit User Interface

The screenshot shows the Ranger Audit User Interface. The top navigation bar includes links for Access Manager, Audit, and Settings, along with a user icon for 'admin'. Below the navigation is a secondary menu with tabs for Access, Admin, Login Sessions, Plugins, Plugin Status, and User Sync. The main content area displays audit logs for the service 'hdp\_kafka' with access type 'consume'. The logs are filtered by start date '05/28/2020'. The table has columns for Policy ID, Event Time, User, Service, Resource, Name / Type, Access Type, Result, Access Enforcer, Client IP, Cluster Name, Event Count, and Tags. All entries show 'Allowed' results for 'ranger-acl' enforcers. The last updated time is listed as '05/28/2020 12:29:25 AM'.

Policy ID	Event Time	User	Service	Resource	Name / Type	Access Type	Result	Access Enforcer	Client IP	Cluster Name	Event Count	Tags
60	05/28/2020 12:30:20 AM	rangertagsync	hdp_kafka	ATLAS_ENTIT... topic	kafka	consume	Allowed	ranger-acl	172.30.12.184	hdp	10	--
60	05/28/2020 12:30:15 AM	rangertagsync	hdp_kafka	ATLAS_ENTIT... topic	kafka	consume	Allowed	ranger-acl	172.30.12.184	hdp	10	--
60	05/28/2020 12:30:10 AM	rangertagsync	hdp_kafka	ATLAS_ENTIT... topic	kafka	consume	Allowed	ranger-acl	172.30.12.184	hdp	10	--
60	05/28/2020 12:30:05 AM	rangertagsync	hdp_kafka	ATLAS_ENTIT... topic	kafka	consume	Allowed	ranger-acl	172.30.12.184	hdp	10	--
60	05/28/2020 12:30:00 AM	rangertagsync	hdp_kafka	ATLAS_ENTIT... topic	kafka	consume	Allowed	ranger-acl	172.30.12.184	hdp	10	--
60	05/28/2020 12:29:55 AM	rangertagsync	hdp_kafka	ATLAS_ENTIT... topic	kafka	consume	Allowed	ranger-acl	172.30.12.184	hdp	10	--
60	05/28/2020 12:29:50 AM	rangertagsync	hdp_kafka	ATLAS_ENTIT... topic	kafka	consume	Allowed	ranger-acl	172.30.12.184	hdp	10	--

# 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 Ranger has support for controlling access to Kafka

# Bibliography

---

The following offer more information on topics discussed in this chapter

- [Cloudera Kerberos Authentication with Kafka](#)
- [Apache Ranger](#)
- [Cloudera Security Training class \(OnDemand\)](#)



## Conclusion

---

Chapter 18

# Course Chapters

---

- Introduction
- Kafka Overview
- Deploying Apache Kafka
- Kafka Command Line Basics
- Using Streams Messaging Manager (SMM)
- Kafka Java API Basics
- Improving Availability through Replication
- Improving Application Scalability
- Improving Application Reliability
- Analyzing Kafka Clusters with SMM
- Monitoring Kafka
- Managing Kafka
- Message Structure, Format, and Versioning
- Improving Application Performance
- Improving Kafka Service Performance
- Securing the Kafka 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
- Understand fundamental resource considerations for planning Kafka deployments

# 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*