Cassandra and Kafka Support on AWS/EC2

Cloudurable

Support around Cassandra and Kafka running in EC2

CLOUDURABLE

Cassandra / Kafka Support in EC2/AWS

Kafka Introduction

Kafka messaging

What is Kafka?

- Distributed Streaming Platform
  - Publish and Subscribe to streams of records
  - Fault tolerant storage
  - Process records as they occur

Kafka Usage

- Build real-time streaming data pipe-lines
  - Enable in-memory microservices (actors, Akka, Vert.x, Qbit)
- Build real-time streaming applications that react to streams
  - Real-time data analytics
  - Transform, react, aggregate, join real-time data flows

Kafka Use Cases

- Metrics / KPIs gathering
  - Aggregate statistics from many sources
- Even Sourcing
  - Used with microservices (in-memory) and actor systems
- Commit Log
  - External commit log for distributed systems. Replicated data between nodes, re-sync for nodes to restore state
- Real-time data analytics, Stream Processing, Log Aggregation, Messaging, Click-stream tracking, Audit trail, etc.

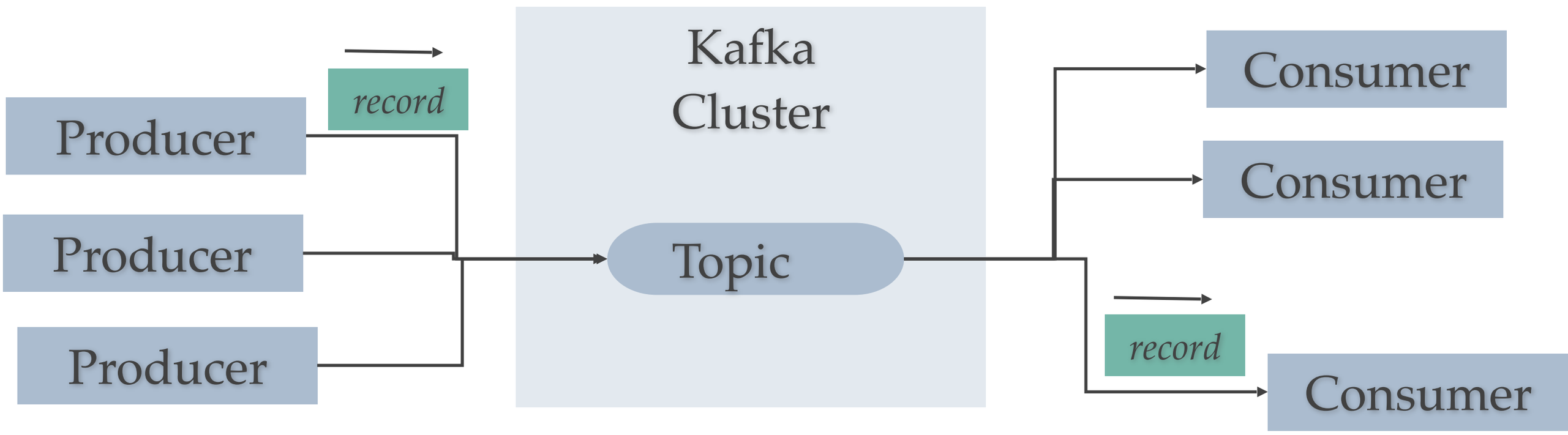
Who uses Kafka?

- ***LinkedIn***: Activity data and operational metrics
- ***Twitter***: Uses it as part of Storm – stream processing infrastructure
- ***Square***: Kafka as bus to move all system events to various Square data centers (logs, custom events, metrics, an so on). Outputs to Splunk, Graphite, Esper-like alerting systems
- Spotify, Uber, Tumbler, Goldman Sachs, PayPal, Box, Cisco, CloudFlare, DataDog, LucidWorks, MailChimp, NetFlix, etc.

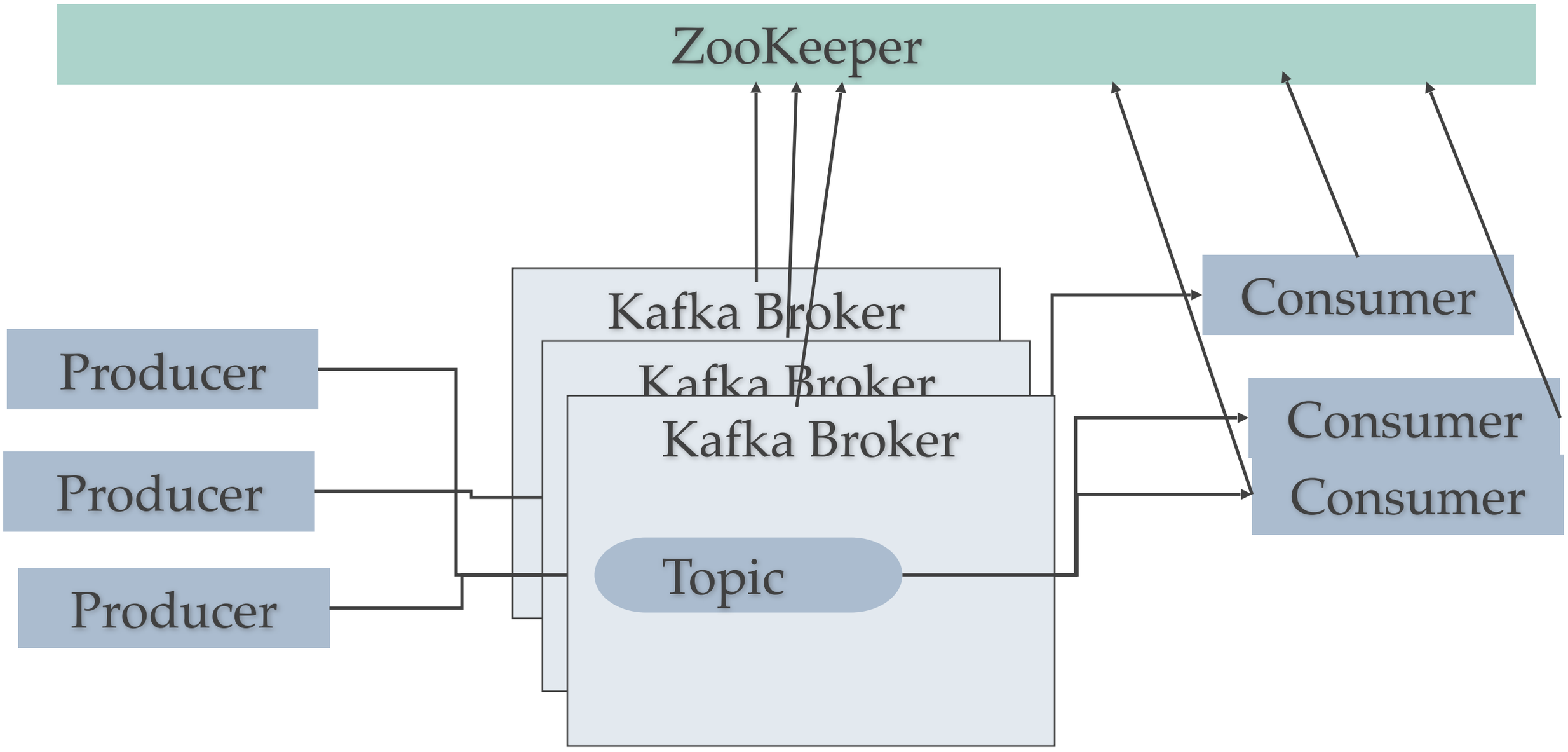
Kafka Fundamentals

- ***Records*** have a ***key***, ***value*** and ***timestamp***
- ***Topic*** a stream of records
  - ***Log*** topic storage on disk
  - Partition / Segments (parts of Topic Log)
- ***Producer*** API to produce a streams or records
- ***Consumer*** API to consume a stream of records
- ***Broker***: Cluster of Kafka servers running in cluster form broker. Consists on many processes on many servers
- ***ZooKeeper***: Does coordination of broker and consumers. Consistent file system for configuration information and leadership election

Kafka: Topics, Producers, and Consumers



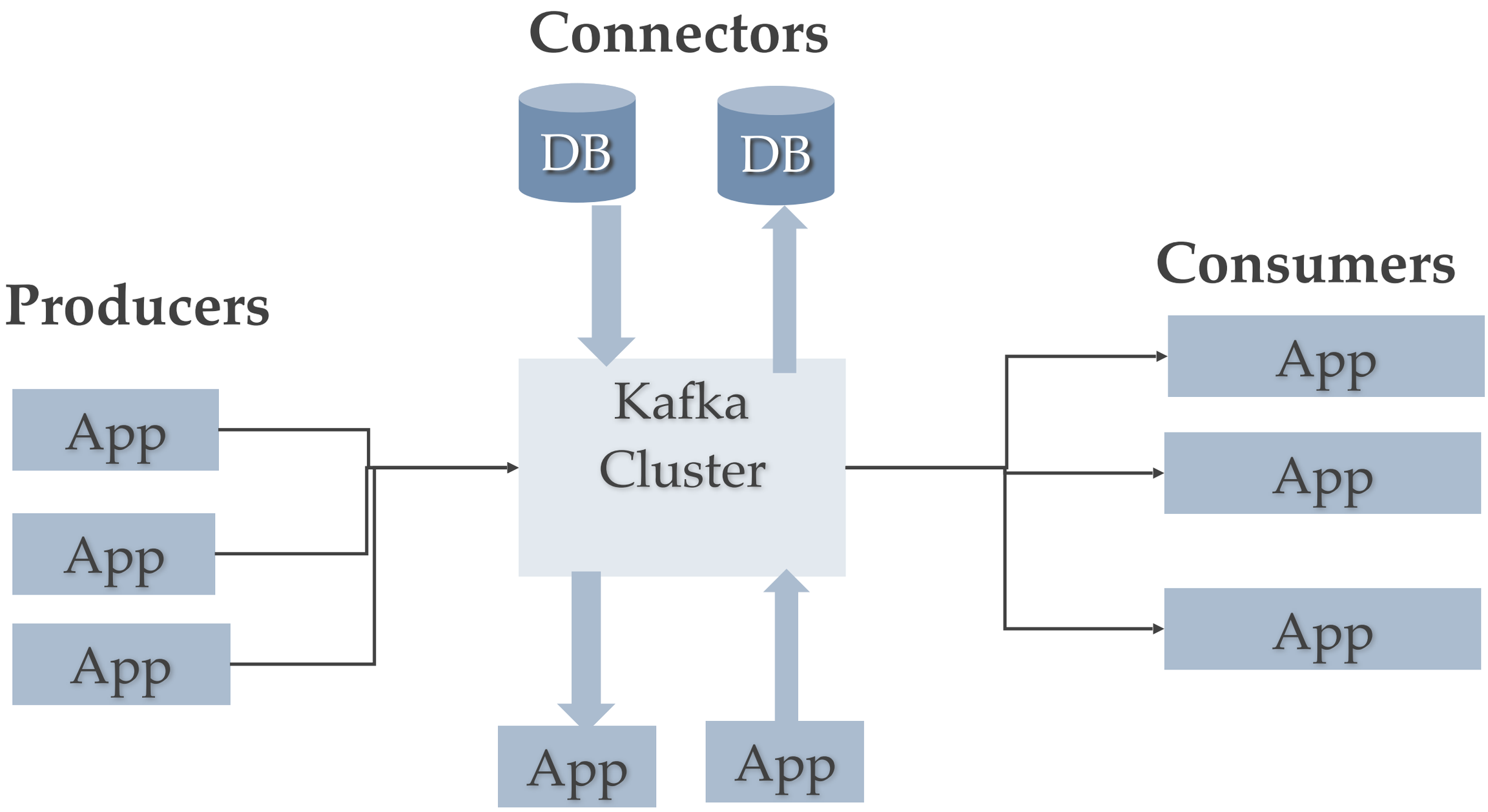
ZooKeeper does coordination for Kafka Consumer and Kafka Cluster



Kafka Extensions

- ***Streams*** API to transform, aggregate, process records from a stream and produce derivative streams
- ***Connector*** API reusable producers and consumers (e.g., stream of changes from DynamoDB)

Kafka Connectors and Streams



Kafka Polyglot clients / Wire protocol

- Kafka communication from clients and servers wire protocol over TCP protocol
- Protocol versioned
- Maintains backwards compatibility
- Many languages supported

Topics and Logs

- Topic is a stream of records
- Topics stored in log
- Log broken up into partitions and segments
- Topics is a category or stream name
- Topics are pub/sub
  - Can have zero or many consumers (subscribers)
- Topics are broken up into partitions for speed and size

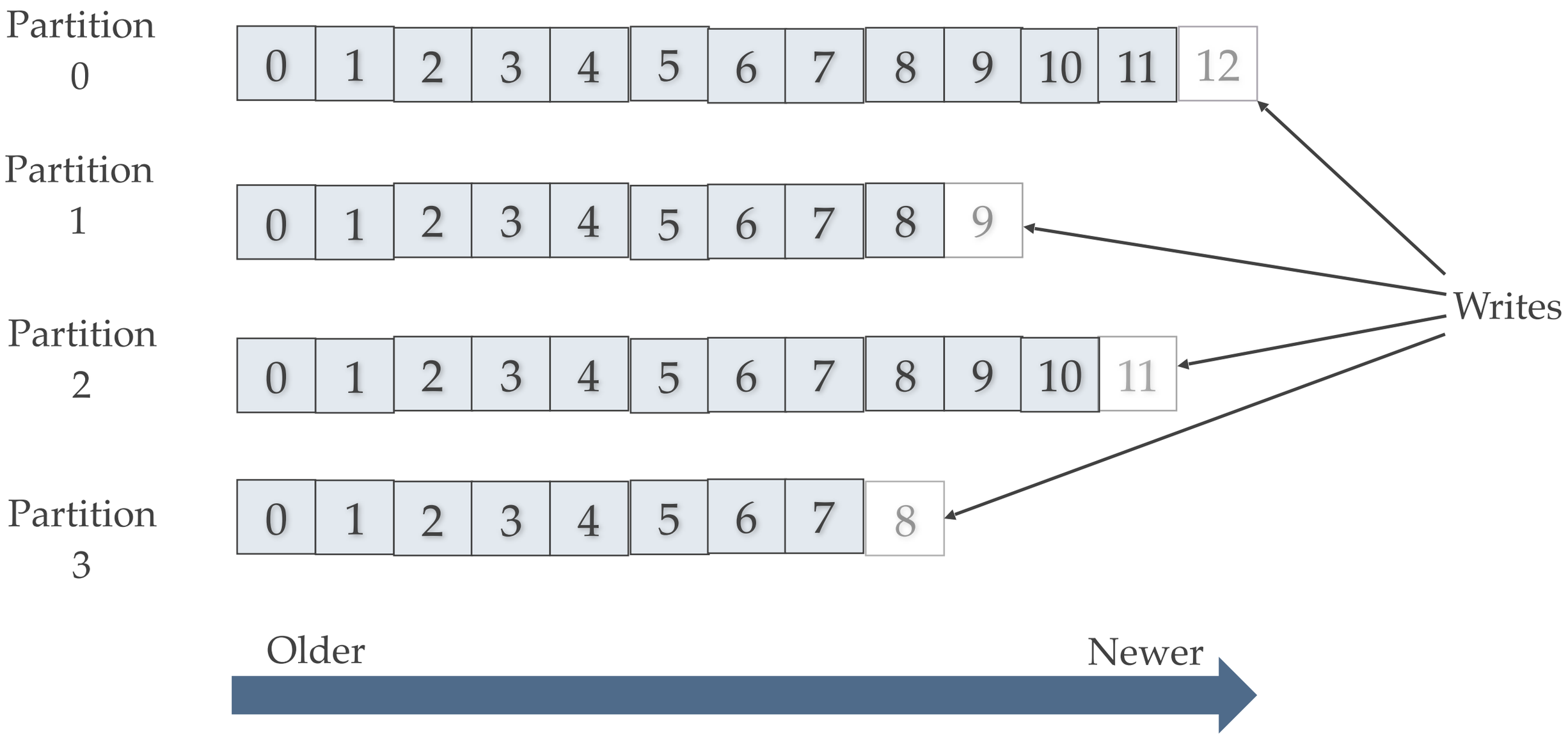
Topic Partitions

- ***Topics*** are broken up into ***partitions***
- ***Partitions*** are decided usually by key of record
  - Key of record determines which partition
- ***Partitions*** are used to scale Kafka across many servers
  - Record sent to correct partition by key
- ***Partitions*** are used to facilitate parallel consumers
  - Records are consumed in parallel up to the number of partitions

Partition Log

- ***Partition*** is ordered, immutable sequence of records that is continually appended to—a structured commit ***log***
- Records in partitions are assigned ***sequential id*** number called the *offset*
- Offset identifies each record within the partition
- ***Topic Partitions*** allow Kafka log to scale beyond a size that will fit on a single server
  - Topic partition must fit on servers that host it, but topic can span many partitions hosted by many servers
- Topic Partitions are unit of ***parallelism*** - each consumer in a consumer group can work on one partition at a time

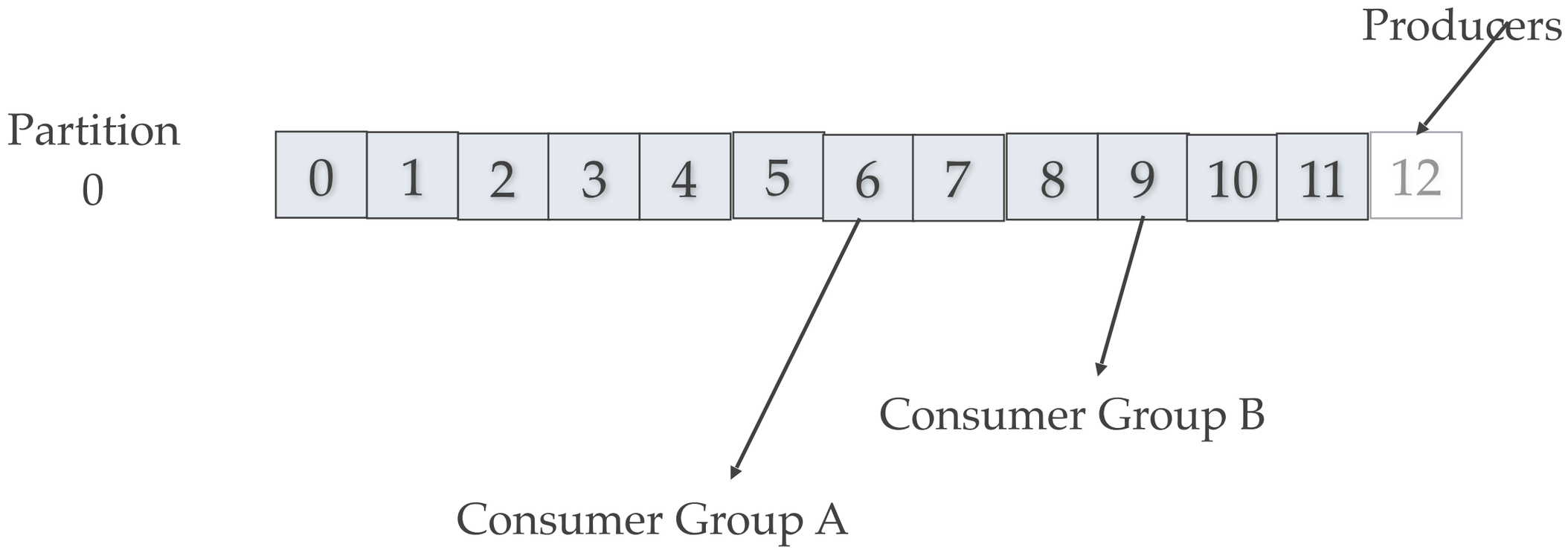
Kafka Topic Partitions Layout



Kafka Record retention

- Kafka cluster retains all published records
  - Time based – configurable retention period
  - Size based
  - Compaction
- Retention policy of three days or two weeks or a month
- It is available for consumption until discarded by time, size or compaction
- Consumption speed not impacted by size

Kafka Consumers / Producers



Consumers remember offset where they left off.

Consumers groups each have their own offset.

Kafka Partition Distribution

- Each partition has ***leader server*** and zero or more ***follower servers***
  - ***Leader*** handles all read and write requests for partition
  - ***Followers*** replicate leader, and take over if leader dies
    - Used for parallel consumer handling within a group
- Partitions of log are distributed over the servers in the Kafka cluster with each server handling data and requests for a share of partitions
- Each partition can be replicated across a configurable number of Kafka servers
  - Used for fault tolerance

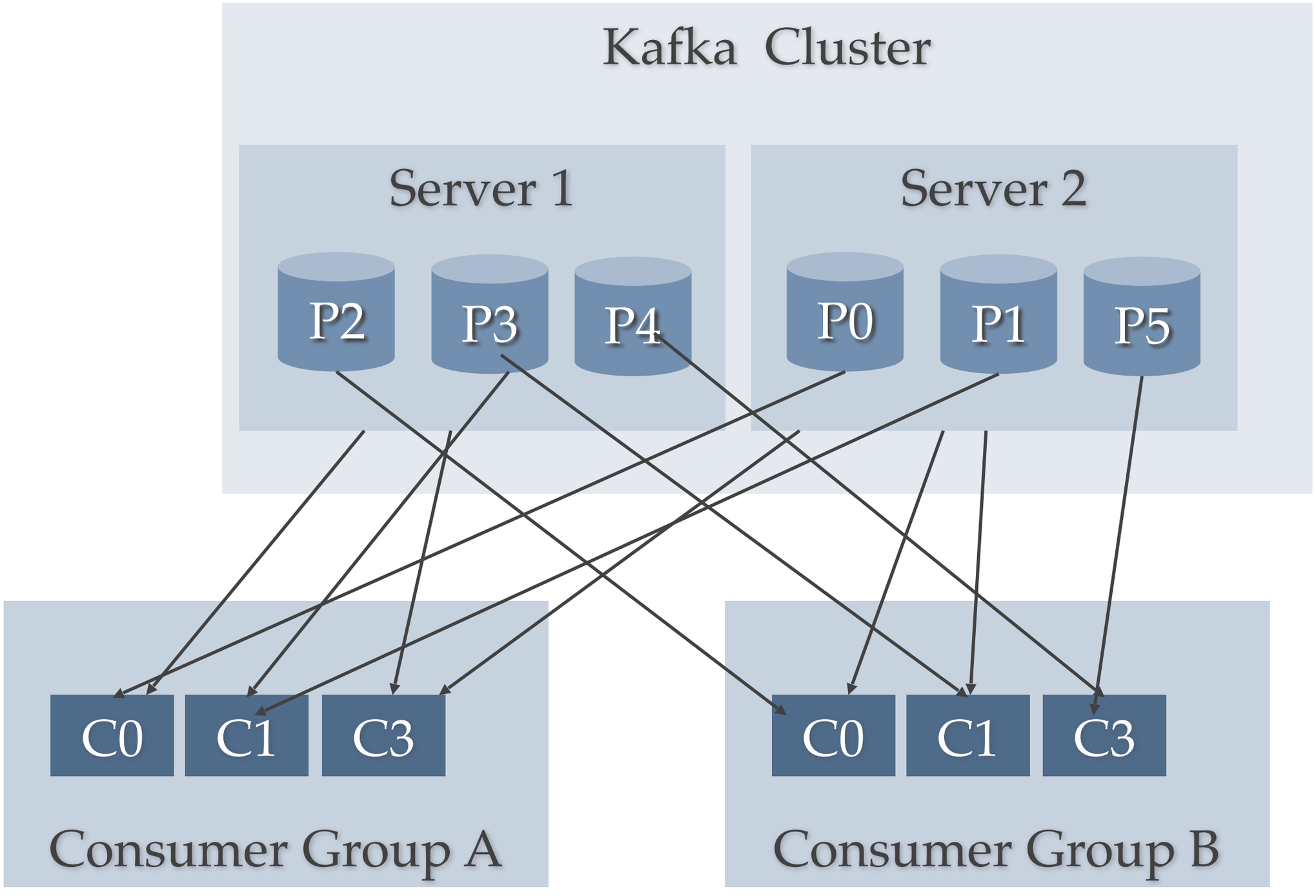
Kafka Producers

- ***Producers*** send records to topics
- ***Producer*** picks which partition to send record to per topic
  - Can be done in a ***round-robin***
  - Can be based on priority
  - Typically based on ***key*** of ***record***
- Important: *Producer picks partition*

Kafka Consumers

- Consumers are grouped into a ***Consumer Group***
  - ***Consumer group*** has a unique name
  - Each ***consumer group*** is a subscriber
  - Each ***consumer group*** maintains its own offset
  - Multiple subscribers = multiple consumer groups
- ***A Record*** is delivered to one ***Consumer*** in a ***Consumer Group***
- Each consumer in consumer groups takes records and only one consumer in group gets same record
- Consumers in Consumer Group ***load balance record consumption***

2 server Kafka cluster hosting 4 partitions (P0-P5)



Kafka Consumer Consumption

- Kafka ***Consumer*** consumption divides partitions over consumer instances
  - Each Consumer is exclusive consumer of a "fair share" of partitions
  - Consumer membership in group is handled by the Kafka protocol dynamically
    - If new Consumers join Consumer group they get share of partitions
    - If Consumer dies, its partitions are split among remaining live Consumers in group
- Order is only guaranteed within a single partition
- Since ***records*** are typically stored ***by key into a partition*** then order per partition is sufficient for most use cases

Kafka vs JMS Messaging

- It is a bit like both Queues and Topics in JMS
- Kafka is a queue system per consumer in consumer group so load balancing like JMS queue
- Kafka is a topic/pub/sub by offering Consumer Groups which act like subscriptions
  - Broadcast to multiple consumer groups
- By design Kafka is better suited for scale due to partition topic log
- Also by moving location in log to client/consumer side of equation instead of the broker, less tracking required by Broker
- Handles parallel consumers better

Kafka scalable message storage

- Kafka acts as a good storage system for records/messages
- Records written to Kafka topics are persisted to disk and replicated to other servers for fault-tolerance
- Kafka Producers can wait on acknowledgement
  - Write not complete until fully replicated
- Kafka disk structures scales well
  - Writing in large streaming batches is fast
- Clients/Consumers control read position (offset)
  - Kafka acts like high-speed file system for commit log storage, replication

Kafka Stream Processing

- Kafka for Stream Processing
  - Kafka enable ***real-time*** processing of streams.
- Kafka supports stream processor
  - Stream processor takes continual streams of records from input topics, performs some processing, transformation, aggregation on input, and produces one or more output streams
- A video player app might take in input streams of videos watched and videos paused, and output a stream of user preferences and gear new video recommendations based on recent user activity or aggregate activity of many users to see what new videos are hot
- Kafka Stream API solves hard problems with out of order records, aggregating across multiple streams, joining data from multiple streams, allowing for stateful computations, and more
- Stream API builds on core Kafka primitives and has a life of its own

Using Kafka Single Node

Run Kafka

- Run ZooKeeper
- Run Kafka Server / Broker
- Create Kafka Topic
- Run producer
- Run consumer

Run ZooKeeper

run-zookeeper.sh

```
#!/usr/bin/env bash
cd ~/kafka-training

kafka/bin/zookeeper-server-start.sh kafka/config/zookeeper.properties &
```

```
rick@Richards-MacBook-Pro-2.local:~/kafka-training
$ ./run-zookeeper.sh
rick@Richards-MacBook-Pro-2.local:~/kafka-training
$ [2017-04-14 17:45:53,408] INFO Accepted socket connection from /0:0:0:0:0:0:0:1:56952 (org.apache.zookeeper.server.NIOServer
CnxnFactory)
[2017-04-14 17:45:53,415] INFO Client attempting to establish new session at /0:0:0:0:0:0:0:1:56952 (org.apache.zookeeper.serv
er.ZooKeeperServer)
[2017-04-14 17:45:53,417] INFO Established session 0x15b6ec06f690014 with negotiated timeout 6000 for client /0:0:0:0:0:0:0:1:
56952 (org.apache.zookeeper.server.ZooKeeperServer)
[2017-04-14 17:45:57,612] INFO Reading configuration from: kafka/config/zookeeper.properties (org.apache.zookeeper.server.quor
um.QuorumPeerConfig)
```

Run Kafka Server

run-kafka.sh

```
#!/usr/bin/env bash
cd ~/kafka-training

kafka/bin/kafka-server-start.sh kafka/config/server.properties
```

```
rick@Richards-MacBook-Pro-2.local:~/kafka-training
$ kafka/bin/kafka-server-start.sh kafka/config/server.properties
[2017-04-14 17:49:09,709] INFO KafkaConfig values:
        advertised.host.name = null
        advertised.listeners = null
        advertised.port = null
        authorizer.class.name =
        auto.create.topics.enable = true
        auto.leader.rebalance.enable = true
        background.threads = 10
        broker.id = 0
```

Create Kafka Topic

create-topic.sh

```
#!/usr/bin/env bash

cd ~/kafka-training

# Create a topic
kafka/bin/kafka-topics.sh --create --zookeeper localhost:2181 \
--replication-factor 1 --partitions 1 --topic my-topic

# List existing topics
kafka/bin/kafka-topics.sh --list --zookeeper localhost:2181
```

Kafka Producer

start-producer-console.sh

```
#!/usr/bin/env bash
cd ~/kafka-training

kafka/bin/kafka-console-producer.sh --broker-list \
localhost:9092 --topic my-topic
```

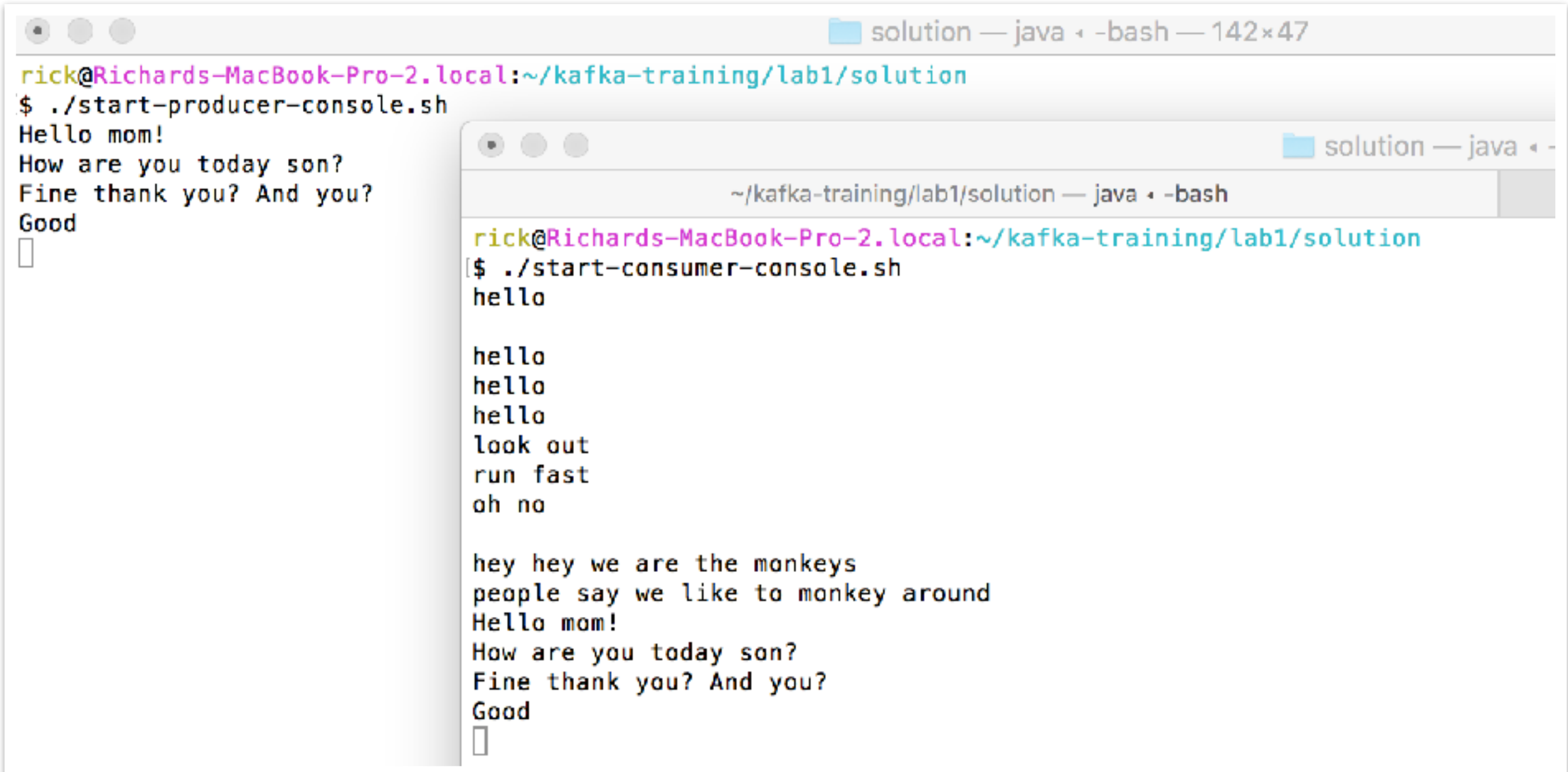
Kafka Consumer

start-consumer-console.sh

```
#!/usr/bin/env bash
cd ~/kafka-training

kafka/bin/kafka-console-consumer.sh --bootstrap-server localhost:9092 \
--topic my-topic --from-beginning
```

Running Kafka Producer and Consumer



Use Kafka to send and receive messages

Lab 1-A Use Kafka

Use single server version of Kafka

Using Kafka Cluster

Running many nodes

- Modify properties files
  - Change port
    - Change Kafka log location
- Start up many Kafka server instances
- Create Replicated Topic

Stay tuned