Providing Durability

Workshop: Apache Kafka Operations

Agenda

- Basic Replication Concepts
- Durability through Intra-Cluster Replication
- Writing Data to Kafka Reliably
- Broker Shutdown and Failures
- Controllers in the Cluster
- The Kafka Log Files
- Offset Management
- Hands-on lab: Investigating the Distributed Log

Basic Replication Concepts

Why Replication?

Each Partition is stored on a Broker

Without replication, if a Broker goes down then:

- Some Partitions will be unavailable
- Permanent data loss could occur

Why would a Broker go down?

- **Controlled shutdown**: rolling restart for code or configuration changes
- **Uncontrolled shutdown**: isolated Broker failure, machine crash

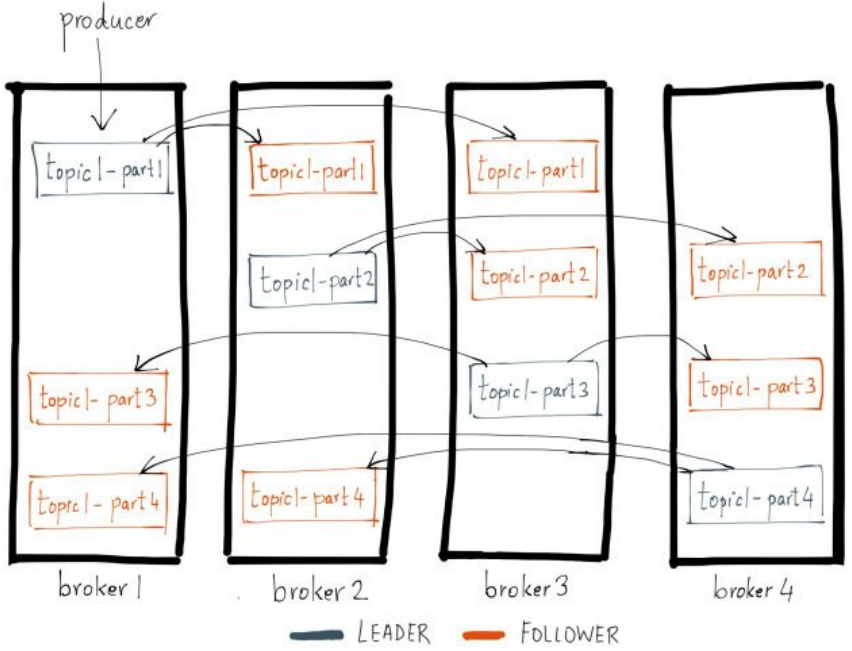
Replication provides higher availability and durability for the cluster

Replicas and Layout

Each Partition can have Replicas

Replicas will be placed on different Brokers

Replicas are spread ***evenly*** among Brokers



Replicas Can Be Rack Aware

Rack-awareness enables replicas to be placed on Brokers in different racks

- **Improves** fault tolerance and increases performance

Useful if deploying Kafka on Amazon EC2 instances across availability zones in the same region.

Replica Configuration

Increase the replication factor for better durability guarantees

For auto-created Topics:

- `default.replication.factor` (Default: 1)
- Configure this parameter in the *server.properties* file on each Broker

For manually-created Topics:

```
$ kafka-topics --create --zookeeper zk_host:2181 --partitions 2
--replication-factor 3 --topic my_topic
```

Durability through Intra-Cluster Replication

Broker Roles for Replication

Brokers have **different roles** that dictate how messages are replicated across those Brokers

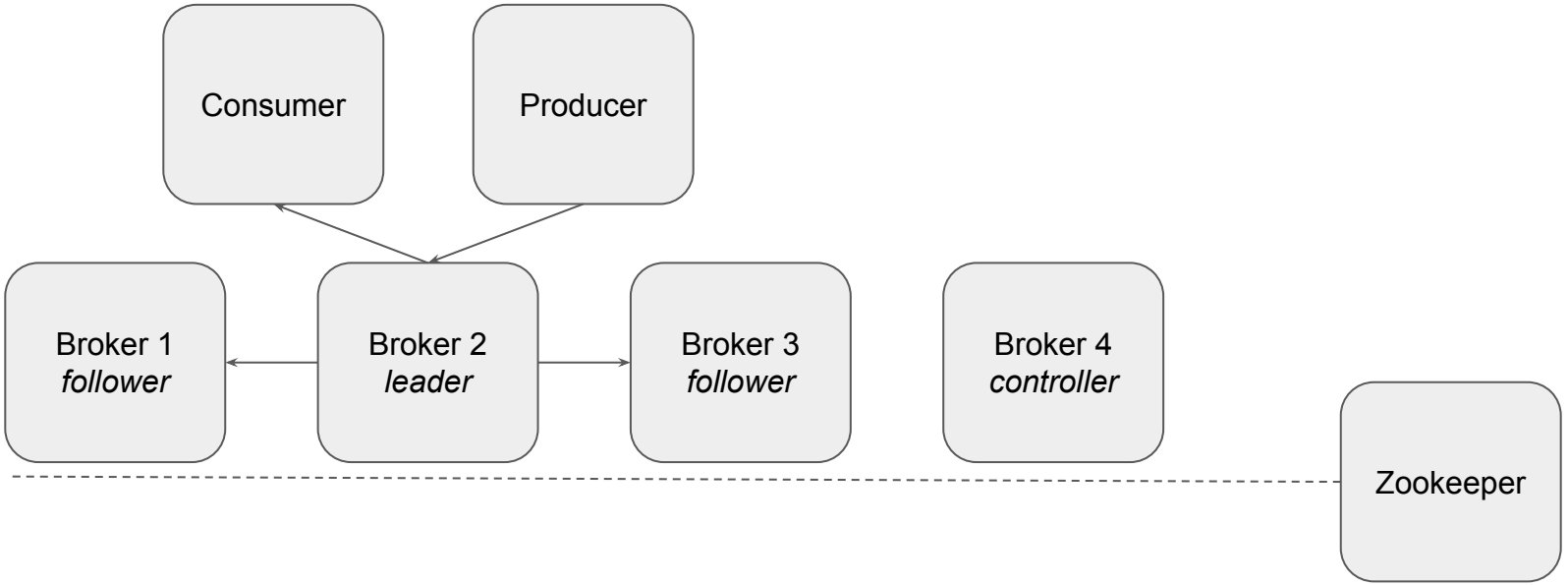
Brokers ensure strongly consistent replicas:

- **One replica** is on the **leader Broker**
- **All Producer** messages **go to the leader**
- The **leader propagates** those messages in order **to the follower Brokers**
- **All Consumers** read messages **from the leader**

One Broker in the cluster is the Controller (singleton)

- Communicates leader and replica information to other Brokers (more on that later)

Replica Leaders and Followers



Replica Leaders and Followers

Leader

- Accepts all writes and reads for a specific Partition
- Manages replicas on other Brokers for a specific Partition
- A new leader will automatically be elected if the current leader is shutdown or fails
- Leader election rate (Meter)
    - `kafka.controller:type=ControllerStats,name=LeaderElectionRateAndTimeMs`

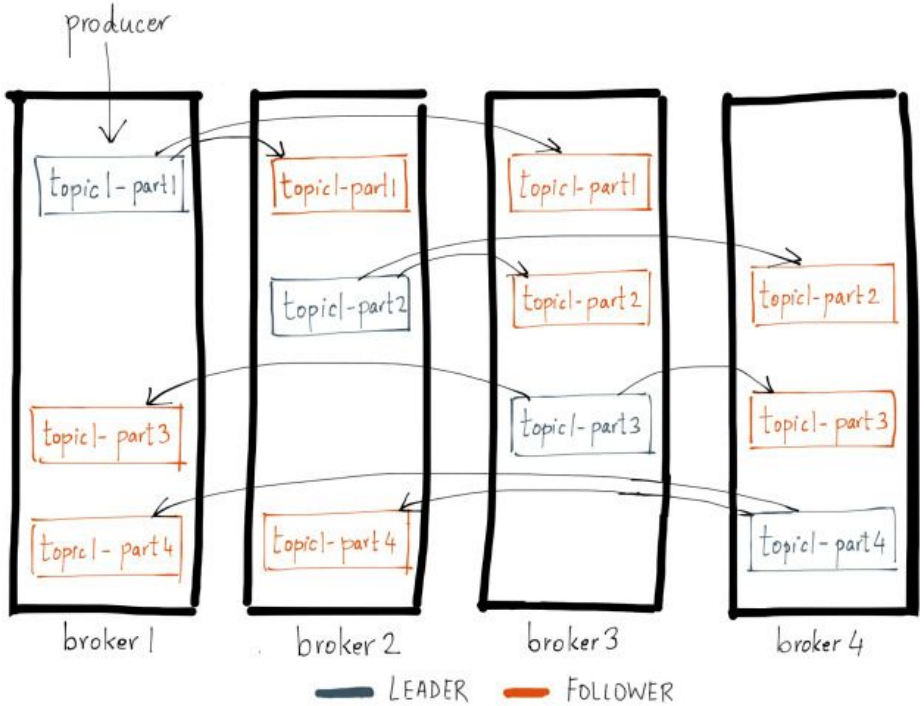
Followers

- Attempt to keep up with the leader
- Provide fault tolerance

Distributed Partition Leaders

Ideally, leaders should be evenly distributed across all Brokers

- Leaders do more work
- Leaders can change on failure



In Sync Replicas (ISR)

In-Sync Replicas (ISRs) are the **subset of followers caught-up to the leader**

A message is committed if it is received by every replica in the ISR list

```
[training@confluent-training-vm ~]$ kafka-topics --zookeeper zookeeper1:2181 --describe --topic i-love-kafka
Topic:i-love-kafka      PartitionCount:3        ReplicationFactor:3     Configs:
        Topic: i-love-kafka     Partition: 0    Leader: 103     Replicas: 103,101,102   Isr: 103,101,102
        Topic: i-love-kafka     Partition: 1    Leader: 101     Replicas: 101,102,103   Isr: 101,102,103
        Topic: i-love-kafka     Partition: 2    Leader: 102     Replicas: 102,103,101   Isr: 102,103,101
```

What Does Committed Really Mean?

Committed means that the messages are received by all the replicas in the ISR

Consumers cannot read (fetch) messages until they are committed

The leader decides when to commit a message

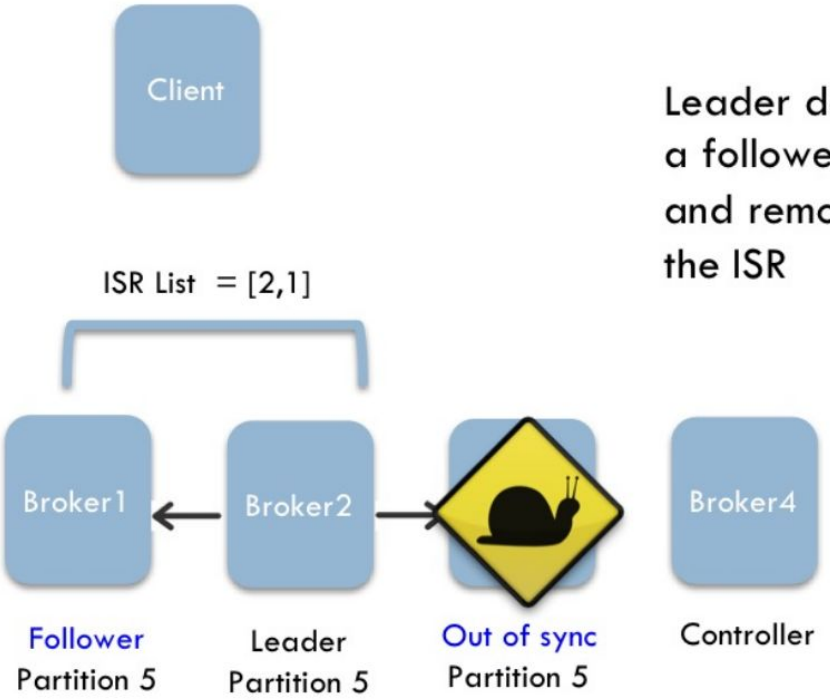
- Committed state is checkpointed to disk

Detecting Slow Replicas

Leader keeps ISR by monitoring the followers lag and removing from the ISR if too high

- Setting is replica.lag.time.max.ms
- This setting is important:
  - Controls the lag between leader and follower replica
  - Too large, and slow replicas will slow down writes
  - Too small, and replicas will drop in and out of ISR

Detecting Slow Replicas



Leader detects that a follower is slow and removes from the ISR

replica.lag.time.max.ms

Monitoring ISR

`UnderReplicatedPartitions`

- `kafka.server:type=ReplicaManager,name=UnderReplicatedPartitions`
- Number of under-replicated Partitions (| ISR | < | all replicas |). Alert if the value is greater than 0 for a long time

`IsrExpandsPerSec`

- `kafka.server:type=ReplicaManager,name=IsrExpandsPerSec`
- When a Broker is brought up after a failure, it starts catching up by reading from the leader. Once it is caught up, it gets added back to the ISR

`IsrShrinksPerSec`

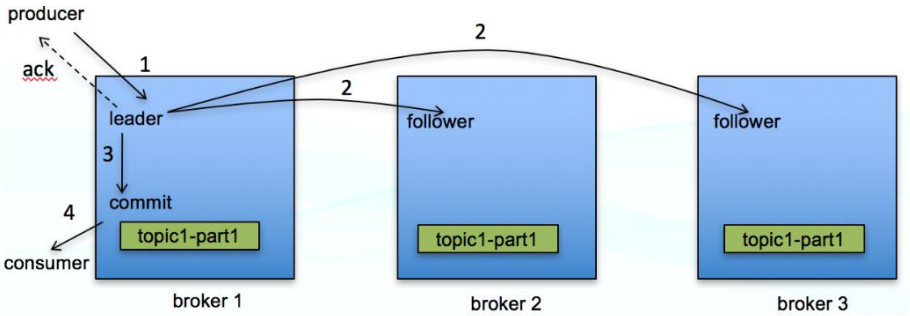
`kafka.server:type=ReplicaManager,name=IsrShrinksPerSec`

Writing Data to Kafka Reliably

Producer Acks

Developers can control message durability with the acks Producer setting

- acks: number of acknowledgments the Producer requires the leader to have received from in-sync replicas before considering a request complete (Default: 1)



When Producer receives ack?	Latency	Durability on failures
No ack (acks=0)	No network delay	Some data loss
Wait for leader (acks=1)	1 network roundtrip	A little data loss
Wait for committed (acks=all)	2 network roundtrips	No data loss

Producer Retries

Developers can configure the retries configuration setting in the Producer code

- retries: how many times the Producer will attempt to retry sending records
  - Only relevant if acks is not 0
- Hides transient failure
- Ensure lost messages are retried rather than just throwing an error
  - retries: number times to retry (Default: 0)
  - retry.backoff.ms: pause added between retries (Default: 100)
  - For example, retry.backoff.ms=100 and retries=600 will retry for for 60 seconds

Preserve Message Send Order

If retries > 0, message ordering could change

To preserve message order, set max.in.flight.requests.per.connection=1 (Default: 5)

- May impact throughput performance because of lack of request pipelining

Broker Shutdown and Failures

Broker Shutdown

Messages that are written into the log files are initially stored in the page cache

- This gives Kafka such good performance
- Periodically the messages are flushed to disk

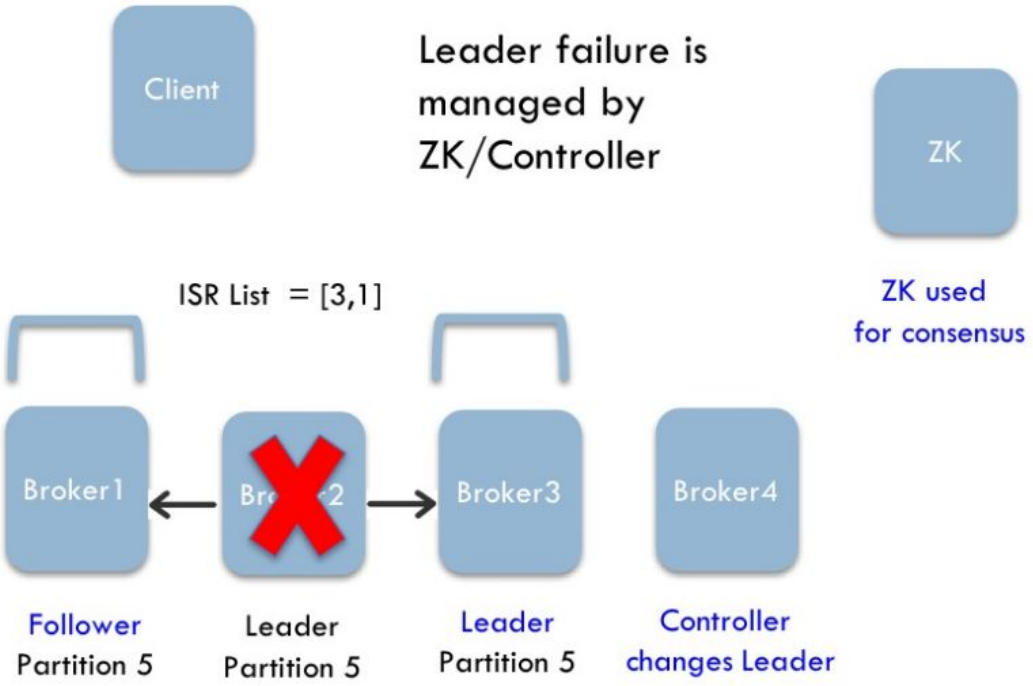
If a Broker is shut down cleanly

- Messages are flushed prior to shutdown

If a Broker is not shut down cleanly

- Some messages may not be flushed to disk yet
  - Topics with replication, on startup the replica recovery process will recover messages
  - Topics without replication, there may be permanent data loss

Leader Failure



- **Requires a new leader to be elected**

Partitions Without Leaders

If a Partition has no active leader, it is not writable or readable

- This may be a transient period if a leader Broker fails
- Producer send will not block, will retry according to retries configuration
- Callback may raise exception NetworkException if retries=0

`OfflinePartitionsCount`

- `kafka.controller:type=KafkaController,Name=OfflinePartitionsCount`
- Number of Partitions that do not have an active leader. Alert if the value is greater than 0

Maintaining the List of In Sync Replicas

Leader maintains a list of ISRs

- Initially, all replicas are in the ISR list
- A message is committed if it is received by every replica in the ISR list

If a follower fails

- It is dropped from the ISR list
- Leader commits using the new ISRs

Maintaining the List of In Sync Replicas

If a leader fails

- A new leader is chosen from the live replicas in the ISR list

Tradeoff

- *n* replicas means we can tolerate *n-1* failures
- Longer latency
- Typically not a big issue within a data center

Guarantees of Replica Recovery

When Broker A failed

- Depending on Producer acks and retries configuration
- Messages that were replicated to followers are not lost
- There may be duplicates due to resends

When the failed Broker A restarts

- A rejoins as a follower
- A catches up to the other replicas by synchronizing from the current leader
- ISR list is restored to [A, B, C]
- B temporarily keeps being the leader
- After a few minutes, the preferred replica (assuming it to be A) becomes the leader

Tradeoff: Availability vs Durability

`unclean.leader.election.enable`

- Determines whether a new leader can be elected even if it is not in sync, if there is no other choice
- This can result in data loss if enabled (Default: false)
- Monitor UncleanLeaderElectionsPerSec
- `kafka.controller:type=ControllerStats,name=UncleanLeaderElectionsPerSec`

`min.insync.replicas (Default: 1)`

- The replica count in the ISR list must not fall below this threshold
  - Otherwise the Producer will receive a NotEnoughReplicas exception
- This provides stronger guarantees when used with acks=all on the Producer

Controllers in the Cluster

Partition State Management

One Broker in the entire cluster is designated as the Controller

- Detects Broker failure/restart via ZooKeeper

Controller action on Broker failure

- Selects a new leader and updates the ISR list
- Persists the new leader and ISR list to ZooKeeper
- Sends the new leader and ISR list changes to all Brokers

Controller action on Broker restart

- Sends leader and ISR list information to the restarted Broker

Partition State Management

If the Controller fails, one of the other Brokers will become the new Controller

Monitoring

- ActiveControllerCount
- kafka.controller:type=KafkaController,name=ActiveControllerCount
- Number of active Controllers on a Broker. Alert if more than one Broker in the cluster shows 1

The Kafka Log Files

Log File Subdirectories

Kafka “log” files are sometimes called “data” files

Each Broker has one or more data directories specified in the server.properties file

- e.g., log.dirs=/var/lib/kafka-a,/var/lib/kafka-b,/var/lib/kafka-c

Each Topic-Partition has a separate subdirectory

- e.g., /var/lib/kafka-a/my_topic-0 for Topic called my_topic

File Types Per Topic-Partition

Each Topic-Partition subdirectory contains four types of files

- Log files hold the messages and metadata, Filename suffix: `.log`
- Index files map the virtual to physical offsets, Filename suffix: `.index`
- Time-based index files using the timestamp of the messages, Filename suffix: `.timeindex`
- Leader epoch and corresponding offset, Filename: `leader-epoch-checkpoint`
- EOS: speed up building a PID → Sequence map in the Broker after restart, Filename suffix: `.snapshot`

Log Files and Segments

A Partition is comprised of one or more segments

- Each .log data file is a segment of the overall Partition

The segment log filename is the offset of the first message in the segment

A segment is closed, and a new one created, when one of the following is exceeded

- log.segment.bytes (Default: 1GB)
- log.roll.ms or, if that value is not present, log.roll.hours (Default: 168 hours)
- log.index.size.max.bytes (Default: 10MB)

Offset Management

Consumer Offset Management

Ordered consumption per Topic Partition

- Just need to keep track of the offset of the last consumed message

Offsets are managed in a Kafka Topic `__consumer_offsets`

Offsets can be committed automatically or manually by the Consumer

Some special consumers store Offsets on custom repositories

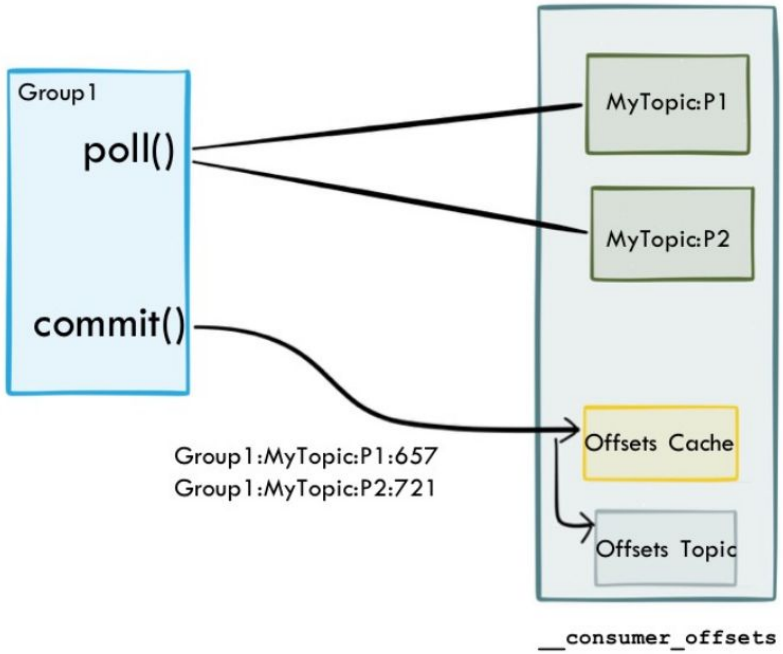
Important Configuration Settings for Offsets

The __consumer_offsets Topic is automatically created when the first Consumer starts consuming from the cluster

Scalability through Partitions: `offsets.topic.num.partitions` (Default: 50)

Resiliency through replication: `offsets.topic.replication.factor` (Default: 3)

Consumers and Offsets (Kafka Topic Storage)



Checking Consumer Offsets

```
$ kafka-consumer-groups --group my-group --describe
--bootstrap-server=broker101:9092,broker102:9092,broker103:9092
```

Check look for the Current Offset and Lag

Hands-on lab: Investigating the Distributed Log