Kafka High Availability

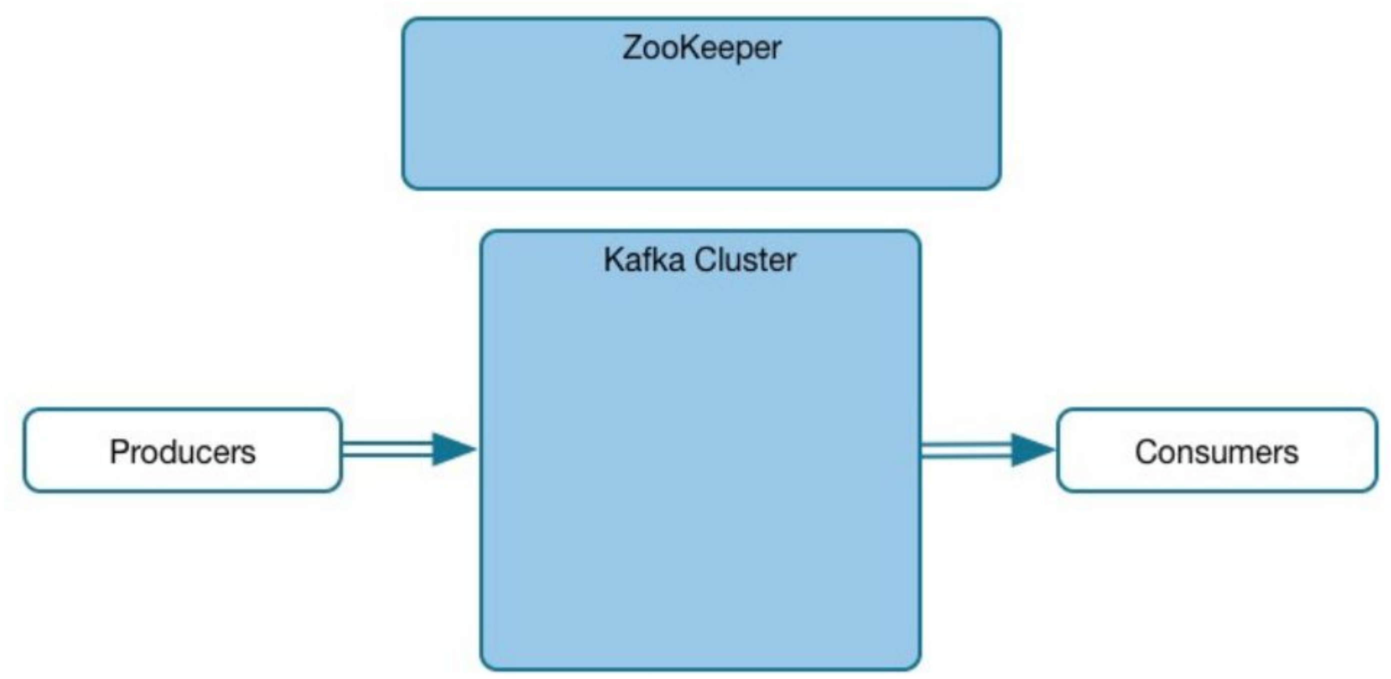
Workshop: Apache Kafka Operations

Agenda

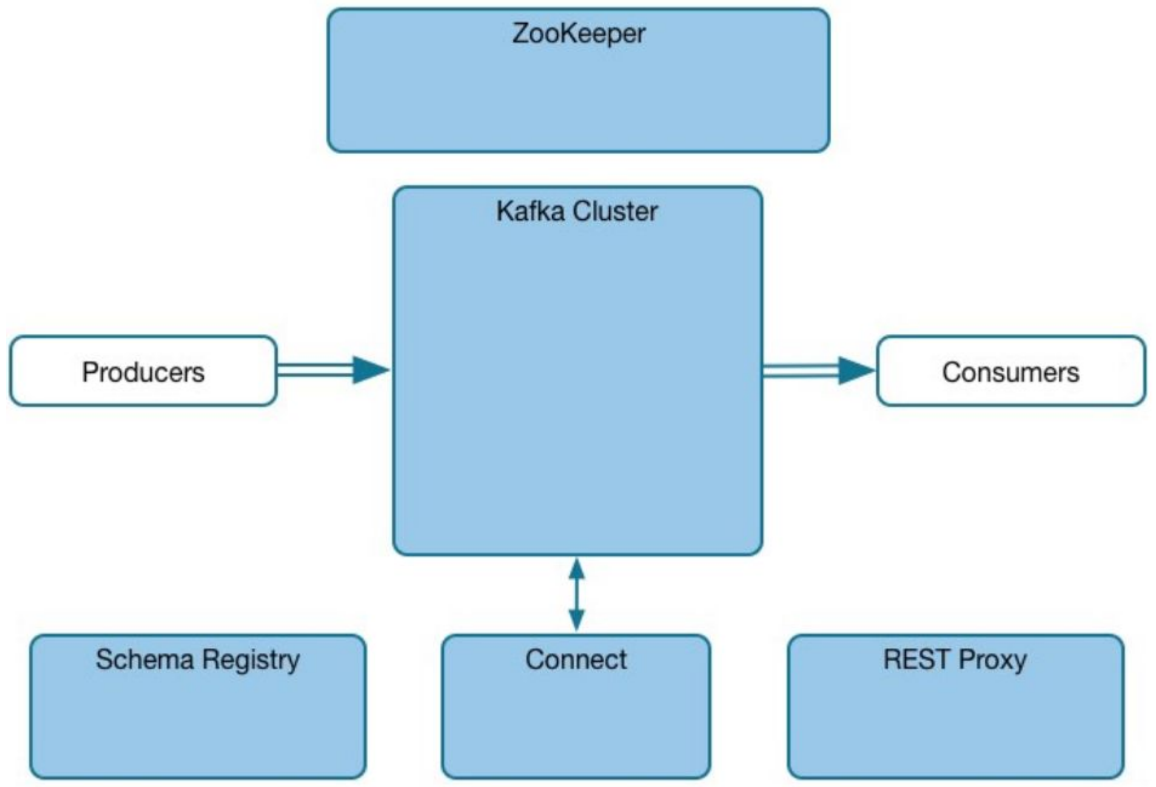
- Typical Kafka Design Model
- Brokers
- Zookeeper
- Kafka Connect
- Schema Registry
- Multiple Data Centers
- Hands-on lab: Prepare Kafka Cluster

Typical Kafka Design Model

Kafka Basic Pub/Sub

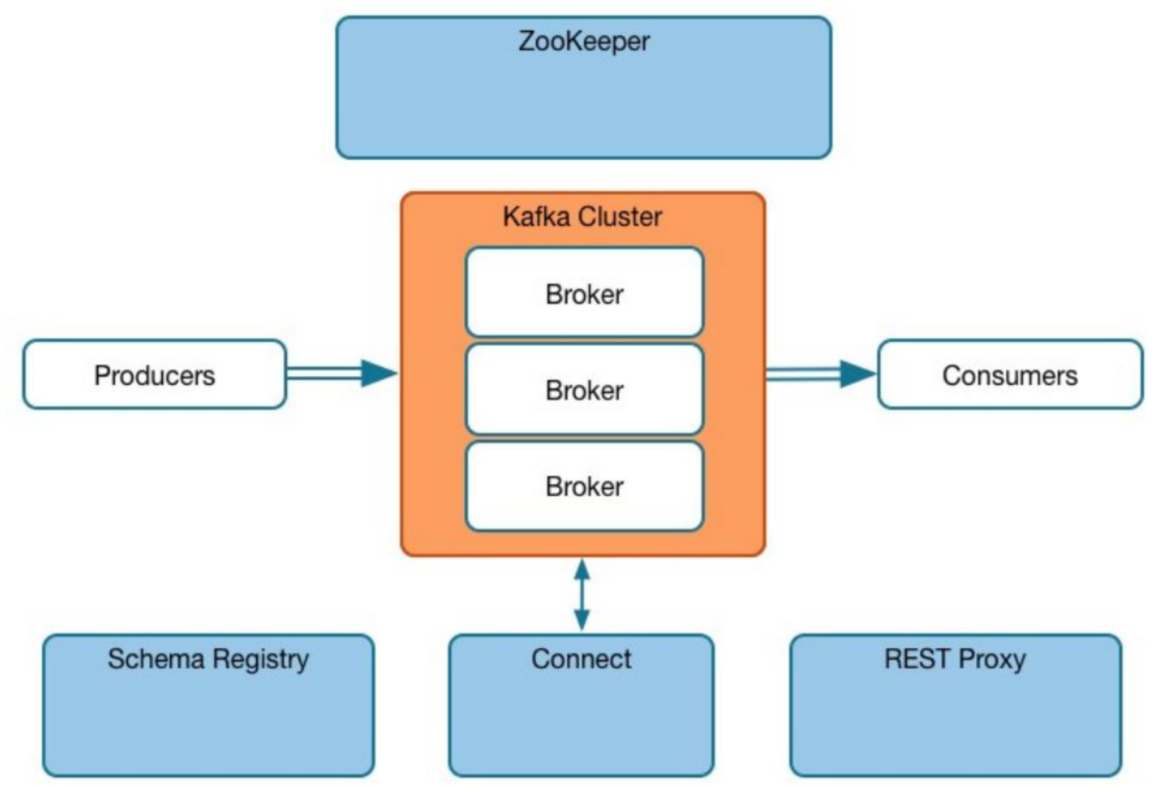


Kafka Reference Architecture



Brokers

Kafka Reference Architecture: Brokers



Broker Design

Run each Broker on a separate server

Deploy Brokers on different servers from ZooKeeper

There should be at least N Brokers to allow replication factor of N

- If the requirement is to replicate data at least 3 times, then deploy at least 3 Brokers

It is possible to use a Virtual IP (VIP) in a load balancer in front of the Brokers

- Brokers might be added or removed from the Kafka cluster
- Clients can use the VIP for the bootstrap servers configuration parameter

Broker System Requirements

Critical resources for a Kafka Broker:

- Disk space and I/O
- Network bandwidth
- RAM (for page cache)

Broker Disk

For lower latency, always use local disk instead of shared storage like NAS or SAN

Support for RAID and JBOD

- Use RAID-10 rather than RAID-5 or RAID-6
- JBOD provides more capacity (no RAID overhead)

Network Considerations

Gigabit Ethernet is sufficient for many applications

- 10Gb Ethernet will help for large installations
  - Particularly for inter-Broker communication

Server RAM Specification

Servers do not need very large amounts of RAM

Kafka Brokers themselves have a relatively small memory footprint

Extra RAM will be used by the operating system for disk caching

- This is the desired behavior for a Kafka Broker

When specifying CPUs, favor more cores over faster cores

- Kafka is heavily multi-threaded

Operating System and Software Requirements

Choose the Linux server operating system you are most familiar with

- Red Hat Enterprise Linux/CentOS and Ubuntu are the most common options

Use the latest release of JDK 1.8

Use the G1 Garbage Collector

Tuning the Java Heap

Java Heap memory is allocated for storing Java objects

By default kafka-server-start configures heap size to 1GB

- Xmx: maximum Java heap size
- Xms: start Java heap size

```
`export KAFKA_HEAP_OPTS="-Xms4G -Xmx4G"`
```

Typical JVM options:

```
-Xms4g -Xmx4g -XX:MetaspaceSize=96m -XX:+UseG1GC -XX:MaxGCPauseMillis=20
-XX:InitiatingHeapOccupancyPercent=35 -XX:G1HeapRegionSize=16M
-XX:MinMetaspaceFreeRatio=50 -XX:MaxMetaspaceFreeRatio=80
```

Tuning the Java Heap

Suggested formula for determining the Broker’s heap size

- `message.max.bytes * # Partitions + log.cleaner.dedupe.buffer.size + 500MB`

Parameter defaults

- `message.max.bytes default is 1MB`
- `log.cleaner.dedupe.buffer.size default is 128MB`

Tuning the Java Heap

Java Heap Size	Deployment Type
1GB (default)	Testing and small production deployments
4GB	Typical production deployments
12GB+	Deployments with very large messages or a very large number of partitions

File System Configuration

File system

- Use XFS or ext4
- Mount with noatime

Increase Open File Handle Limit

Kafka needs a file descriptor for each socket, log segment, index segment, and timeindex segment

```
$ ulimit -n 100000
```

Tuning Virtual Memory Settings

Configure minimal virtual memory swapping

- vm.swappiness=1 (Default: 60)

Configure when unflushed (i.e., “dirty”) memory is flushed to disk

- Increase the frequency of non-blocking background flushes (asynchronous)
- Kafka relies on high disk I/O performance
- vm.dirty_background_ratio=5 (Default: 10)
- Decrease the frequency of blocking flushes (synchronous)
- vm.dirty_ratio=60 (Default: 20)

On RHEL, CentOS, or Ubuntu: set these parameters in /etc/sysctl.conf

Capacity Planning: Brokers

Note: these are rough calculations to determine how many Brokers are needed in the Kafka cluster

- Use this to set initial cluster size, but be prepared to adjust this once you are using Kafka in production

For each of the following resources, calculate the number of Brokers needed

- Storage, Partitions and Network bandwidth

Then take the maximum of the three numbers above

Consider adding more Brokers to this number

- To mitigate against Broker failure, allow for larger data scale than anticipated, etc.

Capacity Planning: Brokers

Storage

```
Number of messages per day * Average size of each message * Retention *
Replication / Usable storage per Broker
```

Partitions

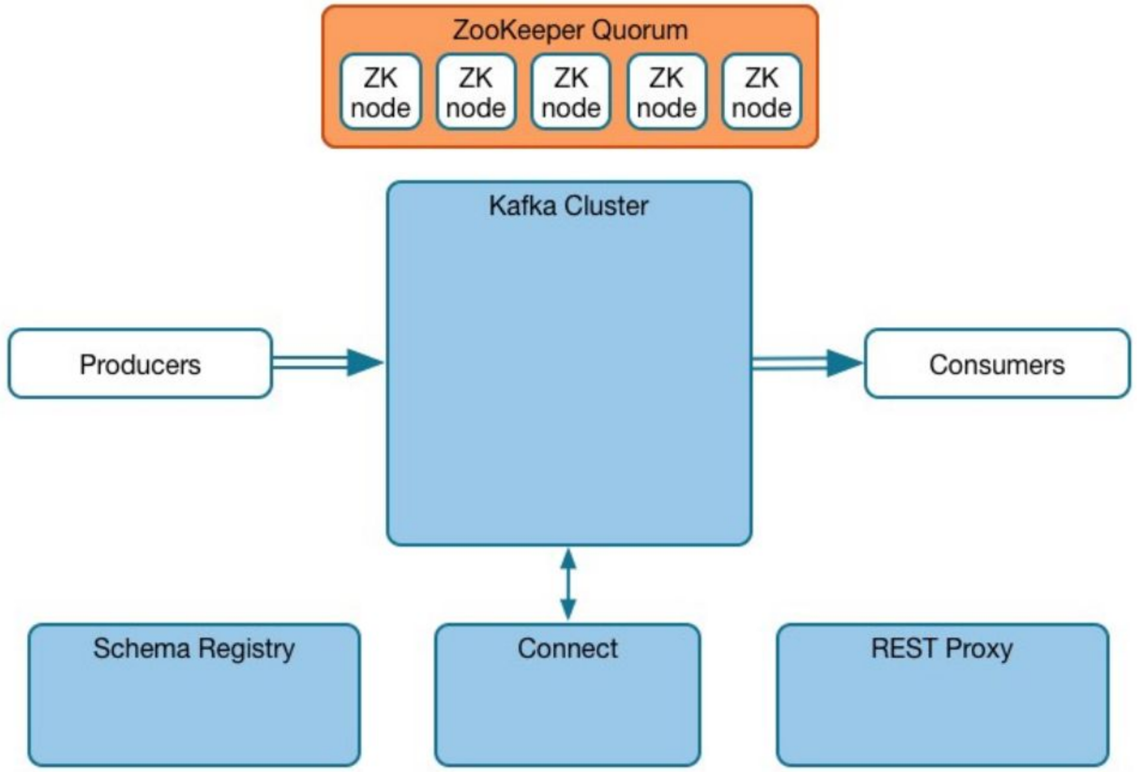
```
Number of Partitions in the cluster * Replication / Number Partitions per
Broker
```

Network bandwidth

```
Number of messages per day * Average size of each message * Number of
Consumers / Network bandwidth per Broker
```

Zookeeper

Kafka Reference Architecture: ZooKeeper



Capacity Planning: ZooKeeper

Deploy ZooKeeper in an ensemble

There should be an odd number of ZooKeeper instances

- ZooKeeper works by Quorum, i.e., majority votes
- 3 nodes allow for one node failure
- 5 nodes allow for two node failures

System Requirements for ZooKeeper

ZooKeeper is a mission-critical system

ZooKeeper has relatively low CPU requirements

It is sensitive to I/O latency

- Use a dedicated disk for its transaction log
- SATA or high-end SSD

Disk Space on ZooKeeper

Monitor disk utilization by ZooKeeper as well as the Kafka cluster

ZooKeeper saves snapshots and transactional log files

- A snapshot is a persistent copy of the znodes
- These files can grow to be quite large
- Configure ZooKeeper for automatic purging as a simple retention policy
- **autopurge.snapRetainCount:** how many of the most recent ZooKeeper snapshots to retain
- **autopurge.purgeInterval:** time interval in hours for which the purge task has to be triggered

Monitoring ZooKeeper

Use the *ruok* command from ZooKeeper’s command-line shell, responds with *imok* if it is running

Metrics to watch:

- Outstanding requests
- Request latency
- Leader, followers
- Number of clients

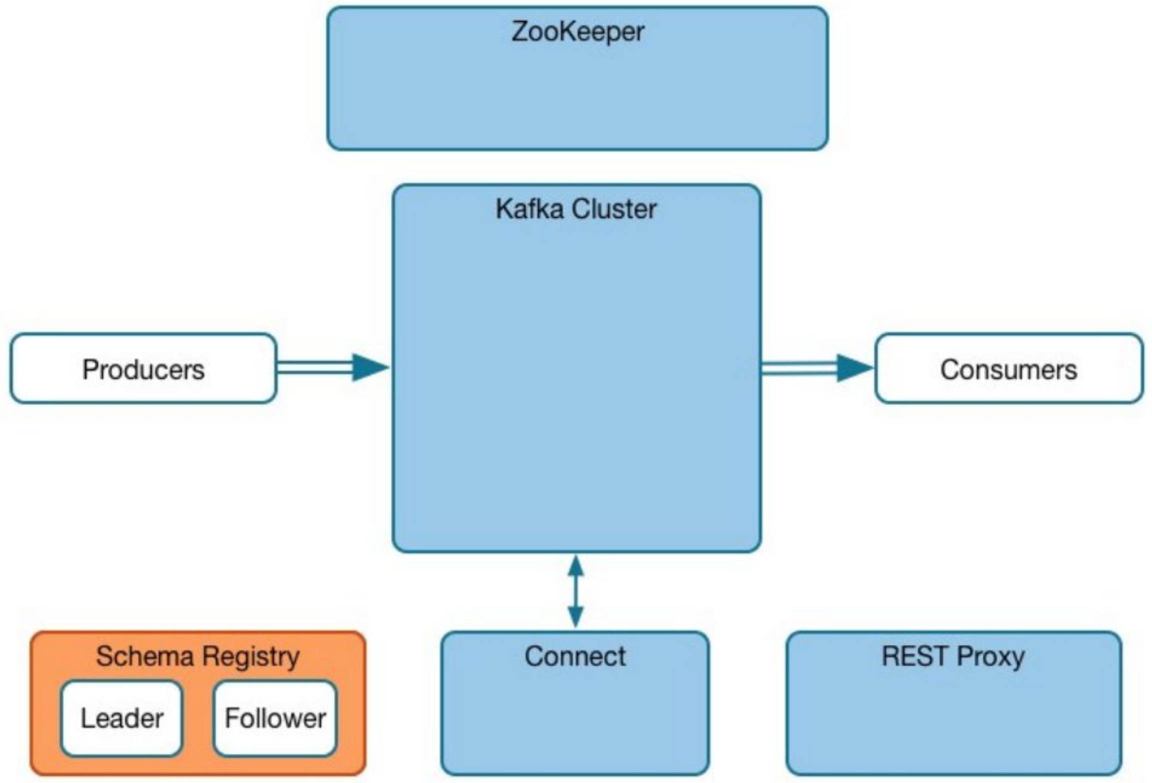
Monitoring ZooKeeper

Watch out for:

- Frequent leader elections
- Number of znodes in the data tree
- Amount of stored state
- Number of open sessions

All ZooKeeper JMX metrics: http://zookeeper.apache.org/doc/r3.4.10/zookeeperJMX.html

Kafka Reference Architecture: Schema Registry



Kafka Schema Registry

Kafka clients can use a data serialization system called Avro

- Data is defined with a self-describing schema
- Supports code generation of data types

Why use a Schema Registry with Avro?

- Sending the Avro schema with each message would be inefficient
- Checks schemas and throws an exception if data does not conform to the schema
- Allows evolution of schemas according to the configured compatibility setting

Deploying Schema Registry

Confluent Open Source includes a Schema Registry

- Stores schemas centrally in a Kafka Topic rather than writing them along with each record
- By default, kafkastore.topic is named _schemas
- Resiliency through replication: `kafkastore.topic.replication.factor (Default: 3)`
- If the number of Brokers in the cluster is less than `kafkastore.topic.replication.factor`
- Auto creation of this Topic will succeed with a replication factor equal to the Broker count
- A warning will be logged in the Schema Registry logfile

Capacity Planning: Schema Registry

Schema Registry needs very few resources

We recommend that you deploy at least two servers for high availability

- Consider using a Virtual IP (VIP)

With multiple servers, Schema Registry users a master-slave architecture

- There is only one master at any given time
- Only the master can respond to write requests
- Slaves forward write requests to the master
- The master or slaves can respond to read requests

Multiple Data Centers

Multiple Data Centers

Multiple data centers (DCs) provide services across different locations

- Disaster Recovery: in case of DC failures
- Geographical Locality: save cross-DC bandwidth

Types of Kafka deployments in multiple DC scenarios

1. Stretched

2. Active/Passive

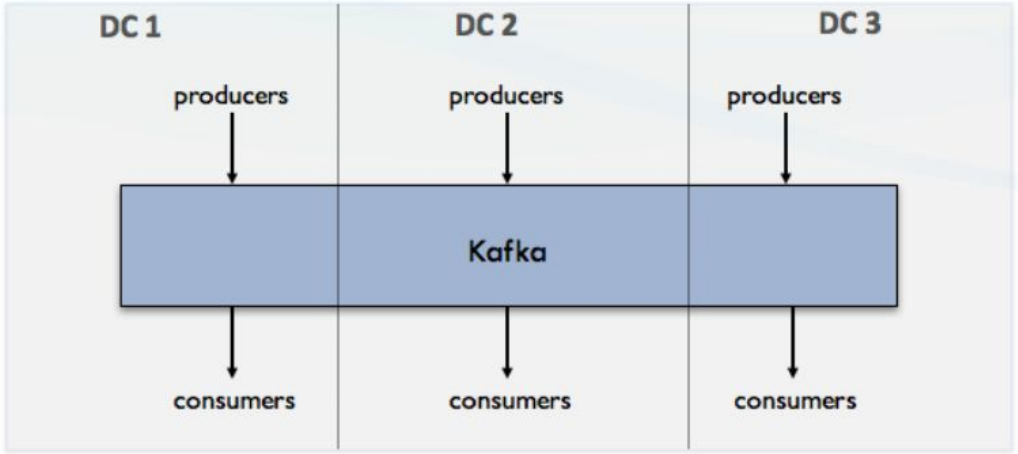
3. Active/Active

Stretched

A “Stretched” deployment is a single logical cluster across DCs that are close to each other

Replication: Kafka’s normal intra-cluster replication mechanisms

- Provides the best guarantees with a simple deployment configuration



Stretched

Deploying Kafka as a stretched cluster

- Requires reliable, low latency network connectivity between DCs
- If in AWS, it can stretch across Availability Zones but should be confined to a single region
- If one site fails, Consumers at the other sites will resume from last committed offset

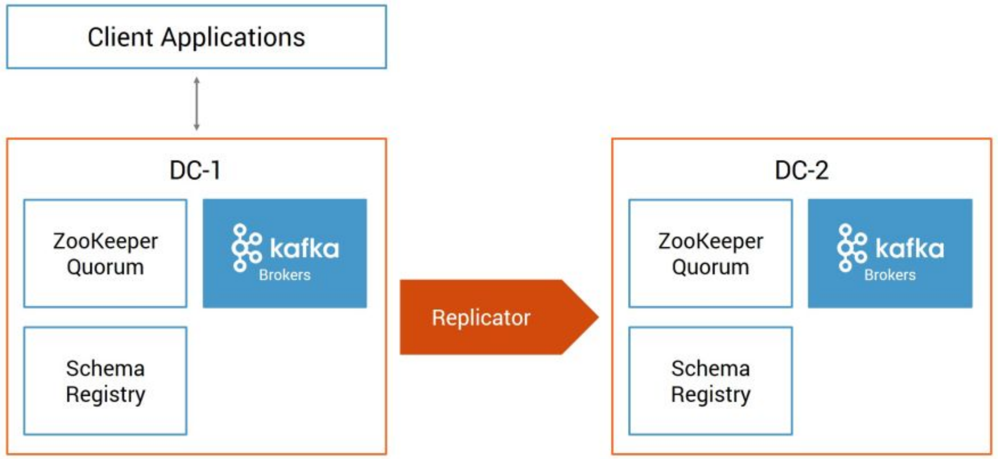
Challenges

- Increased latency for replicating data between sites
- Connectivity between ZooKeeper and Brokers may be up, but connectivity between Brokers in different sites may be down, which will break replication

Active/Passive

An "Active/Passive" deployment has two sites operating as independent clusters with their own ZooKeeper instances

Replication: Replicator for one-way replication from the "Active" origin site to the "Passive" destination site



Active/Passive

Deploying Kafka as active/passive

- Asynchronous state such that the destination site always lags the origin site

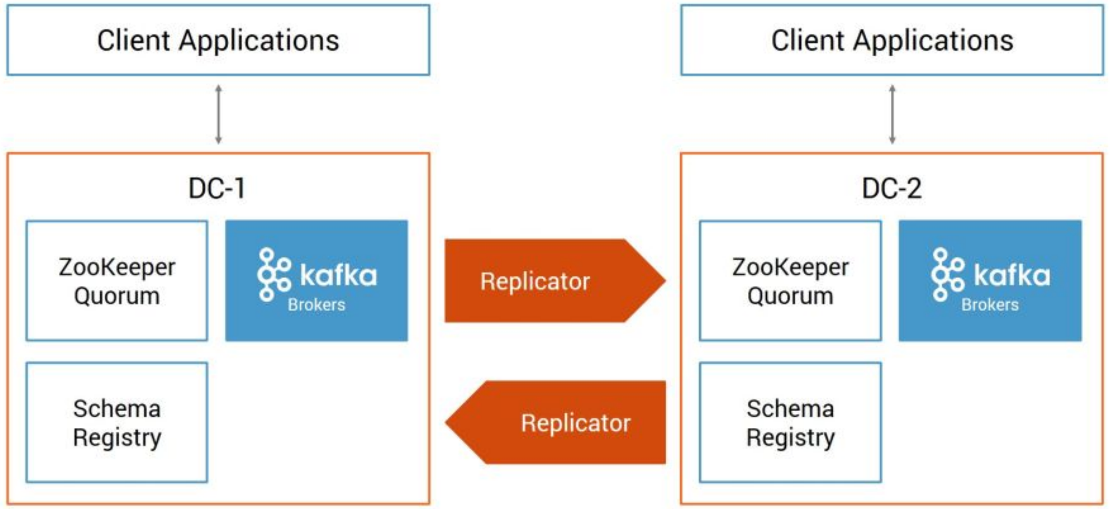
Challenges

- Resources in the destination DC may be underutilized
- If one site fails, Consumers at the other sites will resume from offset defined by `auto.offset.reset`
- smallest: may result in duplicates
- largest: may result in missed messages
- If one site fails, setting up reverse replication is a manual process

Active/Active

Two sites operate as independent clusters with their own ZooKeeper instances

Replication: Replicator for two-way data replication between both sites



Active/Active

Deploying Kafka as active/active

- Producers and Consumers can be in both sites
- Asynchronous state such that the destination site lags the origin site

Cyclic repetition of Topics can be resolved by having unique topic names in each cluster

- Naming original topics to include the DC name (e.g. DC1-topic in DC1, DC2-topic in DC2)
- Replicating topics with new names (e.g. topic in DC1, topic-replica in DC2)

Improving Network Utilization

If network latency is high, increase the TCP socket buffer size in Kafka

- socket.send.buffer.bytes on the origin cluster’s Broker (default: 102400 bytes)
- receive.buffer.bytes on Replicator’s Consumer (default: 65536 bytes)
- Increase corresponding OS socket buffer size

Hands-on lab: Prepare Kafka Cluster