Kafka Operations

Workshop: Apache Kafka Operations

Agenda

- Installing and Running Kafka
- Monitoring Kafka
- Basic Cluster Management
- Log Retention and Compaction
- Hands-on lab: Kafka Administrative Tools

Installing and Running Kafka

Available Formats

Confluent provides Kafka in different formats

- Available as deb, RPM, Zip archive, tarball

Java 7 or Java 8 is required

If you are running on a Mac, use the Zip or Tar archive

Running Kafka on Windows may prove problematic

Running the Kafka Broker

Start ZooKeeper before starting the Kafka cluster

A Kafka Broker can be started and stopped as follows

```
$ /usr/bin/kafka-server-start /etc/kafka/server.properties
$ /usr/bin/kafka-server-stop
```

Running Kafka as a Non-root User

A Kafka Broker can be run as root, or from a non-root account

- In general, it is good practice to run it as a non-root user

If you run as a non-root user, ensure that the directories where Kafka stores its files are writable by that user

- log.dirs: configure the data files directory in the server.properties file (Default: /var/lib/kafka)
- LOG_DIR: configure the log4j files directory by exporting the environment variable (Default: /var/log/kafka)
- Sometimes the term log is used interchangeably for the data file and log file, but these are different!

Client and Broker Version Compatibility

Newer Brokers

- Clients can communicate with Brokers running newer (or same) versions of Kafka
  - Example: a Kafka 0.10.0 Producer can communicate with a Kafka 0.10.1 Broker

Older Brokers

- Clients can communicate with Brokers running older versions of Kafka
  - Example: a Kafka 0.10.2 Producer can communicate with a Kafka 0.10.1 Broker

Caveats

- Clients must be running at least Kafka 0.10.2
- Brokers must be running at least Kafka 0.10.0
- Some features may not be available

Configuring the Cluster Properties

Kafka's default configuration will work for most environments

You can modify available settings at different levels

- Brokers: modify /etc/kafka/server.properties before starting the Broker
- Topics: use kafka-configs command at time of Topic creation or afterwards (more on that later)
- Producers, Consumers: configure in the client code

Check the 'Configurations' section of the Kafka documentation

- http://kafka.apache.org/documentation.html#configuration

Unique Broker Ids

Broker ids need to be unique within a Kafka cluster

Broker ids can be manually configured

- You need to manage Broker id assignments
- broker.id: configurable in the server.properties file
- Default configuration line in /etc/kafka/server.properties is broker.id=0
- Modify the Broker id so that it is unique across all Brokers in the cluster

Auto-Generated Broker Ids

Broker ids can also be auto-generated

- ZooKeeper manages sequencing
- Delete or comment out broker.id in the server.properties file
- broker.id.generation.enable: allow auto-generated Broker ids (Default: true)
- reserved.broker.max.id: auto-generated Broker ids start at this number plus 1 (Default: 1000)
- First auto-generated broker.id would be 1001

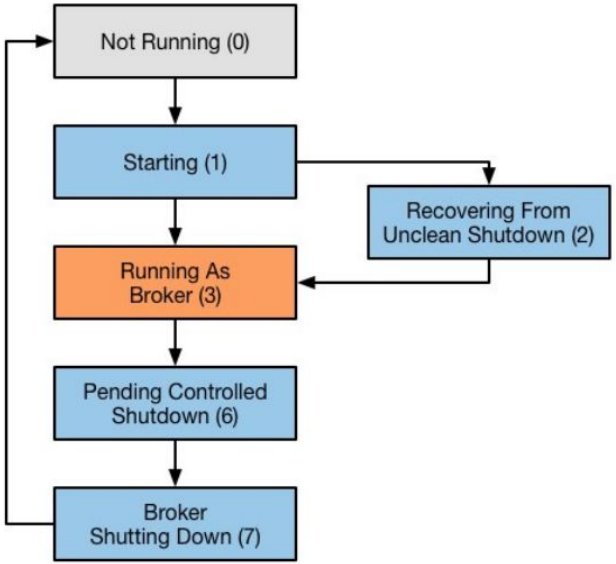
If a Broker were to fail and you don’t want to generate a new Partition plan

- Save the auto-generated Broker id offline
- Assign the same Broker id to the new Broker

Ensuring basic Kafka Broker is Up and Running

Kafka State (Gauge)

- `kafka.server:type=KafkaServer,name=BrokerState`



Upgrading a Cluster

Kafka supports “rolling upgrade”

- No downtime for end users
- Newer Brokers are compatible with clients running older Kafka versions

http://kafka.apache.org/documentation.html#upgrade

Monitoring Kafka

Monitoring Your Kafka Deployments

You can use Confluent Control Center to

- Optimize performance
- Identify potential problems before they happen
- Troubleshoot issues

What else to monitor

- Kafka logs
- Kafka metrics (particularly if not using Confluent Control Center)
- System logs
- System resource utilization

Important log4j Files

By default, the log4j log files are written to /var/log/kafka

- server.log: Broker configuration properties and transactions
- controller.log : all Broker failures and actions taken because of them
- state-change.log: every decision Broker has received from the Controller
- log-cleaner.log : compaction activities
- kafka-authorizer.log : (if enabled) requests being authorized
- kafka-request.log : (if enabled) fetch requests from clients

/etc/kafka/log4j.properties: configuration settings for files above

Tools for Collecting Metrics

Kafka has metrics that can be exposed and inspected through clients

- Combination of Yammer and internal Kafka metrics packages

Confluent Control Center has a Metrics Reporter

There are many other options for clients

- JConsole
- Prometheus
- Grafana
- CloudWatch
- DataDog
- etc.

Configuring the Cluster for Monitoring

JMX metrics can be reported if the Broker is started with the JMX_PORT environment variable set:

- export JMX_PORT=9990

Monitoring Kafka at the OS Level

Items to watch:

- CPU utilization
- Number of open file handles
  - Alert at 80% of the limit
- Disk IO
- Remaining disk space
  - Alert at 60% capacity
- Network bytesIn/bytesOut
  - Alert at 60% capacity

Troubleshooting Issues

Look through the logs

- If needed, enable more detailed level of logging in log4j.properties, WARN→TRACE

Check metrics

- General Kafka metrics
- Specific Producers, Consumers, Consumer Groups, Streams
- System resource utilization

Do not troubleshoot problems by just rebooting nodes to see if the problem “goes away”

- A lot happens when a Broker goes offline, e.g. Leader elections, replica movement
- Extra load is put on the other Brokers (CPU, memory, disk utilization)
- Leaders may not be in sync with preferred replicas

Basic Cluster Management

Topic Configuration Overrides

Broker-level configurations set the default for all Partitions on that Broker

- Set Broker configurations in server.properties
- View runtime Broker configurations in server.log

You can optionally configure per-Topic overrides

- Without an explicit Topic override, the Broker configuration value applies

For example, the Topic-level configuration parameter segment.bytes can override the Broker-level configuration parameter log.segment.bytes

Setting Topic Configurations

Set a Topic configuration at time of Topic creation

```
$ kafka-topics --zookeeper zk_host:port --create --topic my_topic \
--partitions 1 --replication-factor 3 --config segment.bytes=1000000
```

Change a Topic configuration later

```
$ kafka-configs --zookeeper zk_host:port --alter --entity-name my_topic \
--entity-type topics --config segment.bytes=1000000
```

Delete a Topic configuration

```
$ kafka-configs --zookeeper zk_host:port --alter --entity-name my_topic \
--entity-type topics --delete-config segment.bytes
```

Viewing Topic Information

- Show the Topic configuration settings

```
$ kafka-configs --zookeeper zk_host:port --describe --entity-name my_topic \
--entity-type topics
Configs for topic 'my_topic' are segment.bytes=1000000
```

- Show the Partition, leader, replica, ISR information

```
$ kafka-topics --zookeeper zk_host:port --describe --topic my_topic
Topic:my_topic PartitionCount:1
ReplicationFactor:3 Configs:segment.bytes=1000000
Topic: my_topic Partition: 0
Leader: 101 Replicas: 101,102,103
Isr: 101,102,103
```

Deleting Topics

Topic deletion is enabled by default on Brokers, since Kafka 1.0 (Confluent 4.0)

- delete.topic.enable (Default: true)

Caveats

- Stop all Producers/Consumers before deleting
- All Brokers must be running for the delete to be successful

```
$ kafka-topics --zookeeper zk_host:2181 --delete --topic my_topic
```

Adding Partitions

Use the kafka-topics command

```
$ kafka-topics --zookeeper zk_host:port --alter --topic my_topic --partitions 40
```

Doesn’t move data from existing Partitions

- This could cause issues for Consumers of keyed messages
- Consider creating a new Topic with the required number of Partitions and copying data to it from the original Topic

Note: Kafka does not support reducing the number of Partitions in a Topic

Log Retention and Compaction

Managing Log File Growth

Logs are not deleted after consumption because there may be multiple Consumers

- Therefore the logs need a retention policy to manage file growth and free up disk space

Configure the retention policy type with `log.cleanup.policy`

- Types: delete (default) or compact
- Both delete and compact can be enabled at the same time
- A Topic can override the Broker configuration with `cleanup.policy`

Delete Policy Use Case

Use Case for delete

- Temporal event data
- Each record is independent
- Goal is to retain more recent data, delete older data

Retention characteristics

- Coarse-grained time-based retention
- Policy deletes data older than a fixed period of time or when the log reaches some predefined size

Delete Policy: Configuration

Retention policy deletes occur by log segment, not by individual messages

The active log segment is not eligible for deletion

Deletes are triggered when:

- The log cleaner thread runs, periodically
- A log segment rolls

Delete Policy: Configuration

Configure the policy, which also has corresponding Topic overrides

- log.retention.ms
  - The time duration to keep a log segment before it is deleted (Default: 7 days)
- log.retention.bytes
  - The amount of data to retain in the log for each Partition in a Topic (Default: -1)
- log.retention.check.interval.ms
  - Frequency that the log cleaner thread runs (Default: 5 minutes)

Delete Policy: Retention Duration

Retention duration is a guaranteed minimum for how long messages are kept

- Some messages older than configured retention duration may not be deleted when the retention time expires
- Log retention time is based on the largest timestamp of the messages in a log segment

For business requirements where data needs to be deleted after a specific amount of time

- Configure log segments to roll frequently
- log.roll.ms (Default: 7 days)
- Reduce this as needed

Compact Policy Use Case

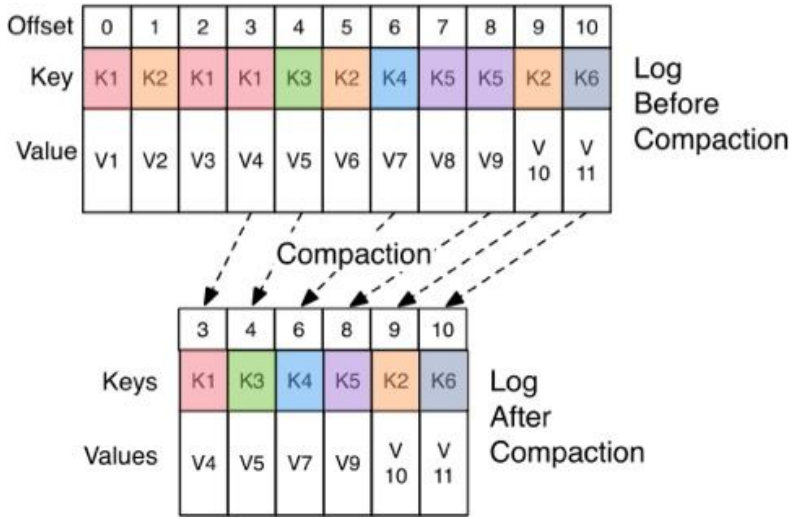
Use Case for compact

- Log of changes to keyed, mutable data
- Records for a given key are related
- Database change capture
- Stateful stream processing
- Event sourcing
- Goal is to retain the most recent keyed value, delete older values for a given key

Compact Policy Use Case

Retention characteristics

- Finer-grained per-record retention providing the last state for each key
- Policy retains at least the last known value for each message key within the log of data for a single Topic Partition
- Requires messages to be published with keys



Log Compaction: Important Configuration Values

log.cleaner.min.cleanable.ratio

- Default is 0.5
- Only trigger log clean if the ratio of dirty/total is larger than this value

log.cleaner.io.max.bytes.per.second

- Default is infinite
- Can be used for throttling

Deleting Keys with Log Compaction

To delete a key, publish a message with that key and a null value

- This delete marker is known as a “tombstone” message
- The tombstone is put into the log file

log.cleaner.delete.retention.ms

- Default 1 day
- The “tombstone” messages are removed after that time
- Consumer should finish consuming the tombstone before that time

Danger of removing a deleted key too soon

- Consumer still assumes the old value with the key

Monitoring Log Compaction

Monitor how frequently and how long Log Compaction attempts to clean the log:

max-dirty-percent

- kafka.log:type=LogCleanerManager,name=max-dirty-percent

cleaner-recopy-percent

- kafka.log:type=LogCleaner,name=cleaner-recopy-percent

max-clean-time-secs

- Last cleaning time
- kafka.log:type=LogCleaner,name=max-clean-time-secs

Hands-on lab: Kafka Administrative Tools