KIP-426: Persist Broker Id to Zookeeper

Status

Current state: *Under Discussion*

Discussion thread: *here*

JIRA: NA

Please keep the discussion on the mailing list rather than commenting on the wiki (wiki discussions get unwieldy fast).

Motivation

KAFKA-1070 introduced auto-generation of broker ids. A broker, if not configured with a broker id, gets one from an auto-incremented sequence in ZooKeeper, then stores it in a file named meta.properties. Upon restart, it reads the id from the meta file and re-registers itself with ZooKeeper using the id.

The mechanism simplifies deployment because the same configuration can be used across all brokers, however, in a large system where disk failure is a norm, the meta file could often get lost, causing a new broker id being allocated. This is problematic because new broker id has no partition assigned to it so it can't do anything, while partitions assigned to the old one lose one replica. This KIP proposes to store additional information into ZooKeeper, and use them to make the process more robust.

Public Interfaces

ZooKeeper Nodes /brokers/hosts/<Host>

This KIP proposes that for each host, a text-formatted JSON object is stored at the permanent ZooKeeper node /brokers/hosts/<Hostname>. The JSON object contains the following keys:

```
{
    "version": 0,
    "broker.id": 10,
}
```

As new keys are added to the JSON in the future, version will be bumped, which helps future Kafka releases reading data generated by older releases. The broker.id is the id of the Kafka broker running on the host.

Proposed Changes

Determining Broker Id on Broker Startup

This KIP aims to improve the procedure described in Motivation above, making it more robust to hardware failure. The new procedure should avoid generating new broker id except when bringing up a new cluster, while maintaining good characteristic of old procedure. Let's take a closer look what information is available:

1. Broker id assigned from command line or server.properties;
2. Broker id stored in meta.properties;
3. (new in this KIP) information stored in ZooKeeper.

Obviously 1 must be the highest priority because this is set explicitly by user. In case 1 is missing, 2 should be higher priority than 3. This is because 2 stays with the data; by reusing a broker id whose assigned partitions are likely to be found locally, by a large chance it minimizes the amount of data transfer during the catch up process. To sum up, this KIP proposes the following procedure:

In case either 1 or 2 is set: first make sure they are consistent if both are set, then use it and bypass the following procedure unconditionally. If the broker fails to register itself in ZooKeeper, an error will be thrown and human intervention is needed. This is exactly the same logic as current.

In case 1 and 2 are both missing, current implementation simply falls back to new id generation. This KIP proposes trying the following procedure instead:

a. If the hostname has been assigned a broker id previously, try registering with the id. If succeeds, use it;
b. (Optional) otherwise look for broker ids that were used previously but are not currently registered:
    i. if there is exactly one, use it;
    ii. if there are more than one, throw error;
    iii. If there are none, fallback to generating a new broker id.

Note: Steps in b) are marked as optional because they are more debatable and takes a lot more effort to implement. If the community has objections to them, they could be removed from this KIP.

Analysis

Step a) is helpful in environments where hostnames are statically assigned. When a broker has hardware failure, and the replacement machine is assigned same hostname, it will get the broker id. In some other environments where the replacement machine has a different hostname, a third-party tool could be used to manipulate the record in ZooKeeper. The paths of the nodes should be part of public interfaces thus stable enough for third-party tools to rely on.

Step b.1) targets the most common case where a single broker is down, thus only one broker id is missing, which is the natural choice for the broker.

Step b.2) targets the following situation: when more than one brokers are down, some of them still has data in their disks, while others lose all their data. It is undesirable for the latter ones to pick up broker id from the prior ones, thus it is better to stop. In an environment where stopped broker can be restarted automatically after some delay, and if only one broker loses its data, it will eventually start up after all other brokers start successfully.

In case multiple brokers lose their data, human intervention is needed. This is because in case replication factor is 3 (which is a very common case) it is already a big problem to lose two brokers' data and deserves human intervention anyway. Theoretically there could be heuristics to do it automatically, but this deserves a separate KIP.

Step b.3) targets the case of new cluster bringing up. Newly started brokers find there is no spare broker ids to re-use, therefore acquire new ones from ZooKeeper.

Persisting (Hostname, Broker Id) Mapping to ZooKeeper

Step a) requires storing the mapping between hostname and broker id. This is done after broker registers itself in ZooKeeper. The section above describes the paths and contents of the ZooKeeper nodes that hold the information.

Look for Missing Broker Ids

The step b) above heavily relies on finding missing broker ids. This KIP proposes using the following way to calculate them:

1. Collect all broker ids by reading all ZooKeeper nodes /brokers/topics/<topic>;
2. Collect all currently using broker ids by listing children of ZooKeeper node /brokers/ids;
3. Taking the difference of the two sets.

Note step 1 is expensive as it needs to read a lot of data from ZooKeeper (~1s for 1k topics with 1ms each). An alternative approach could also be used but is more complicated to implement.

Garbage Collection of ZooKeeper Nodes

As mentioned below, mapping from hostname to broker id will be persisted in ZooKeeper. In an environment where hostnames change frequently, this could create a lot of garbage in the ZooKeeper. Periodically, either a third-party tool or the controller broker will scan the ZooKeeper to remove garbage nodes. A garbage node can be identified in the following logic: for each node /brokers/ids/N, read the hostname h. If there is a node /brokers/hosts/g whose value is N, and g != h, then this is a garbage node.

Compatibility, Deprecation, and Migration Plan

The logic proposed in this KIP tries to work properly for most common cases, however, there are cases where it can't handle properly:

1. Steps in b) not adopted:
    a. In environment where hostname changes;
2. Steps in b) adopted:
    a. In environment where hostname changes, and more than one broker involve in starting process;
    b. Newly joined server steals broker id of another restarting broker;
3. When multiple brokers run on the same host.

Other than these corner cases, there shouldn't be a compatibility issue here. Existing server should able to upgrade to newer version without explicit action; brokers with mixed versions should be able to work together.

~~Rejected~~ Alternatives

The following alternatives are not rejected; they also make sense, but with different trade-offs. This KIP is open to that and could adopt them if the community feel strongly about them.

Have a Boolean Server Configuration to Guard the Change

With this option, the users are given more flexibility to fine tune the startup of the process so that it works best for their own environment. However, by adding yet another configuration about the broker id, it makes the logic more complicated thus more difficult to analyze and reason. Considering the changes in the KIP are executed infrequently, such addition to the complexity doesn't worth the effort.

Read ZooKeeper Before Falling Back to meta.properties

From a different perspective, the ZooKeeper is more reliable than local disk file, thus arguably should be the source of truth. The open question is what key should be used when persisting the broker ids. Logically, if the broker has a reliable source for the key, it can also use the reliable source to store the broker id.

Hostname, as proposed in this KIP, was used as the key, but with the assumption that it is only reliable in certain environments. A counterexample is Amazon EC2, where the default hostname is generated from IP address, while IP address changes across reboots. Another one is when people running multiple test brokers on one host.

If we take this approach eventually, one can add a server configuration flag about what key to use. A user can specify the hostname, or MAC address, etc. as the key, which gives them sufficient flexibility to tailor the startup procedure for their own environment.

Another Way of Looking for Missing Broker Id

The approach proposed in this KIP is expensive and is linear to the number of topics, which probably works for thousands of topic but is difficult to scale even further.

Another approach would be letting controller broker to watch the children of ZooKeeper path /brokers/ids. Whenever one id is gone, it creates a node at another ZooKeeper path, e.g. /brokers/missing_ids/<id>. This scales better, but adds more logic to the controller.