KIP-856: KRaft Disk Failure Recovery

Status

Current state: *Under Discussion*

Discussion thread: https://lists.apache.org/thread/ytv0t18cplwwwqcp77h6vry7on378jzj

JIRA: Unable to render Jira issues macro, execution error.

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

Motivation

KIP-595 introduced KRaft topic partitions. These are partitions with replicas that can achieve consensus on the Kafka log without relying on the Controller or ZK. The KRaft Controllers in KIP-631 use one of these topic partitions (called cluster metadata topic partition) to order operations on the cluster, commit them to disk and replicate them to other controllers and brokers.

The cluster metadata partition is stored in the metadata.log.dir. If this directory doesn't contain a valid meta.properties file the Kafka node fails to start. The user can create a valid meta.properties file by executing the format command of the kafka-storage tool. The format only creates a meta.properties if it doesn't exists.

If the metadata.log.dir was prevously formatted, it was lost because of a disk failure and the user runs the format commmand of the kafka-storage tool then it is possible for the cluster metadata partition to establish a leader with inconsistency or data loss. To safely reformat a metadata.log.dir the user needs to make sure that the new voter node has a superset of the previous voter's on-disk state. This solution to disk failure is manual and error prone.

This KIP describes a protocol for extending KIP-595 and KIP-630 so that the Kafka cluster can detect disk failures on the minority of the nodes in the cluster metadata topic partition. This KIP also describe the mechanism for the user to programmatically replace the failed voters in a way that is safe and available.

Key Terms

These are the definition of some important terms that are used through the document.

Voters: A voter is any replica that can transition to the candidate state and to the leader state. Voters are required to have an ID and UUID. Each replica keeps their own set of voters that are part of the topic partition. For a replica, the voters are the replica ID and UUID is in its own voter set. A candidate needs to get votes from the majority of its own voter set before is can become the leader of an epoch. When a voter becomes a leader it will use its voter set to determine when an offset has been committed.

Observers: An observer is any replica that is not in the voter set. This is because they have an ID and UUID which is not in the voter set or they don't have an ID or UUID.

Proposed Changes

This KIP is inspired by Chapter 4 of Consensus: Bridging Theory and Practice [2]. The description of this KIP makes the assumption that the reader is familiar with the references enumerated at the bottom of this page.

Log Directory Storage UUID

Each log directory of a Kafka node will have a storage UUID. This UUID will be generated once and persisted as part of the log directory's meta.properties. A replica (controller and broker) may host multiple topic partitions in multiple log directories.

There are two cases when a storage UUID will be generated:

1. The kafka-storage CLI tool will generate a storage UUID and meta.properties file for every log directory and the metadata log directory.
2. The meta.properties in the log directory doesn't contain a storage UUID. The meta.properties file will be atomically replace with a new meta.properties file that contains a generated storage UUID.

Replicas in a KRaft topic partition that are voter or can become voters will be identified by their replica ID and storage UUID.

In this KIP the storage UUID is also called ReplicaUuid, CandidateUuid, VoterUuid and VotedUuid depending on the context.

Voter Changes

Adding Voters

Voters are added to the cluster metadata partition by sending an AddVoter RPC to the leader. For safety Kafka will only allow one voter change operation at a time. If there are any pending voter change operations the leader will wait for them to finish. If there are no pending voter change operations the leader will write a AddVoterRecord to the log and immediately update its in-memory quorum state to include this voter as part of the quorum. Any replica that replicates and reads this AddVoterRecord will update their in-memory voter set to include this new voter. Voters will not wait for these records to get committed before updating their voter set.

Once the AddVoterRecord operation has been committed by the majority of the new voter set, the leader can respond to the AddVoter RPC and process new voter change operations.

Removing Voters

Voter are removed from the cluster metadata partition by sending a RemoveVoter RPC to the leader. This works similar to adding a voter. If there are no pending voter change operations the leader will append the RemoveVoterRecord to the log and immediately update its voter set to the new configuration.

Once the RemoveVoterRecord operation has been committed by the majority of the new voter set, the leader can respond to the RPC. If the removed voters is the leader, the leader will resign from the quorum when the RemoveVoterRecord has been committed. To allow this operation to be committed and for the leader to resign the followers will continue to fetch from the leader even if the leader is not part of the new voter set. In KRaft leader election is triggered when the voter hasn't received a successful response in the fetch timeout.

Bootstrapping

The replica id and endpoint for all of the voters is described and persisted in the configuration controller.quorum.voters. An example value for this configuration is `1@localhost:9092,2@localhost:9093,3@localhost:9094` .

Leader of the KRaft topic partition will not allow the AddVoter RPCs to add replica IDs that are not describe in the configuration and it would not allow the RemoveVoter RPCs to completely remove replica IDs that are described in the configuration. This means that the replica UUID can change by using sequence of AddVoter and RemoveVoter but the client cannot add new replica id or remove a replica id from the set of voters.

When the leader of the topic partition first discovers the UUID of all the voters through the KRaft RPCs, for example Fetch and Vote, it will write an AddVoterRecord to the cluster metadata log for each voter described in the controller.quorum.voters. All of these AddVoterRecord must be contained in one control record batch. Putting them all in one record batch guarantees that they all get persisted, replicated and committed atomically.

The KRaft leader will not perform writes from the state machine (active controller) or client until is has written to the log an AddVoterRecord for every replica id in the `controller.quorum.voters` configuration and those control records have been replicated to all of the voters.

If the cluster metadata topic partition contains an AddVoterRecords for a replica id that is not enumerated in controller.quorum.voters then the replica will fail to start and shutdown.

Leader Election

It is possible for the leader to write an AddVoterRecord to the log and replicate it to some of the voters in the new configuration. If the leader fails before this record has been replicated to the new voter it is possible that a new leader cannot be elected. This is because in KRaft voters reject vote requests from replicas that are not in the voter set. This check will be removed and replicas will be able to grant their vote when the candidate is not in the voter set or the voting replica is not in the voter set. The candidate must still have a log with an end offset and epoch that is greater than or equal to the voter's log before winning the vote.

High Watermark

As describe in KIP-595, the high-watermark will be calculated using the log end offset of the majority of the voters. When a replica is removed or added it is possible for the high-watermark to decrease. The leader will not allow the high-watermark to decrease and will guarantee that is is monotonically increasing for both the state machines and the remote replicas.

With this KIP it is possible for the leader to not be part of the voter set when the replica removed is the leader. In this case the leader will continue to handle Fetch and FetchSnapshot requests as normal but it will not count itself when computing the high-watermark.

Snapshots

The snapshot generation code needs to be extended to include these new KRaft specific control records for AddVoterRecord and RemoveVoterRecord. Before this KIP the snapshot didn't include any KRaft generated control records.

Internal Listener

The KRaft implementation and protocol describe in KIP-595 and KIP-630 never read from the log or snapshot. This KIP requires the KRaft implementation now read uncommitted data from log and snapshot to discover the voter set. This also means that the KRaft implementation needs to handle this uncommitted state getting truncated and reverted.

Disk Failure Recovery Scenario

To better illustrate this feature this section describes how Kafka will detect a disk failure and how the administrator can use the features in this KIP to recover from the disk failure.

Let's assume that the cluster is configured to use `controller.quorum.voters` and the value is `1@host1:9092,2@host2:9092,3@host3:9094`. The voter set is defined by the tuples (1, UUID1), (2, UUID2) and (3, UUID3). The first element of the tuples is the replica id. The second element of the tuples is the storage uuid. The storage uuid is automatically generated once by the node when persisting the meta.properties for a log directory.

At this point the operator replaces a failed disk for replica 3. This means that when replica 3 starts it will generate a new storage uuid (UUID3') and meta.properties. Replica 3 will discover the partition leader using `controller.quorum.voters` and sends UUID3' in the Fetch and FetchSnapshot requests. The leader state for voters will be (1, UUID1), (2, UUID2) and (3, UUID3) with the observers being (3, UUID3').

This state can be discovered by a client by using the DescribeQuorum RPC. The client can now decide to add replica (3, UUID3') to the set of voters using the AddVoter RPC.

When the AddVoter RPC succeeds the voters will be (1, UUID1), (2, UUID2), (3, UUID3) and (3, UUID3'). The client can now remove the failed disk from the voter set by using the RemoveVoter RPC.

When the RemoveVoter RPC succeeds the voters will be (1, UUID1), (2, UUID2), and (3, UUID3'). At this point the Kafka cluster has successfully recover from a disk failure in a controller node.

Public Interfaces

Configuration

Not configuration changes are needed for this KIP.

Log and Snapshot Control Records

Two new control records will be added to the log and snapshot of a KRaft partition.

LeaderChangeMessage

Add an optional VoterUuid to Voter. This change is not needed for correctness but it is nice to have for tracing and debugging.

```
{
  "type": "data",
  "name": "LeaderChangeMessage",
  "validVersions": "0-1",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "Version", "type": "int16", "versions": "0+",
      "about": "The version of the leader change message" },
    { "name": "LeaderId", "type": "int32", "versions": "0+", "entityType": "brokerId",
      "about": "The ID of the newly elected leader" },
    { "name": "Voters", "type": "[]Voter", "versions": "0+",
      "about": "The set of voters in the quorum for this epoch" },
    { "name": "GrantingVoters", "type": "[]Voter", "versions": "0+",
      "about": "The voters who voted for the leader at the time of election" }
  ],
  "commonStructs": [
    { "name": "Voter", "versions": "0+", "fields": [
      { "name": "VoterId", "type": "int32", "versions": "0+" },
      { "name": "VoterUuid", "type": "uuid", "versions": "1+" }
    ]}
  ]
}
```

AddVoterRecord

A control record for instructing the replicas to add a new voter to the topic partition. This record can exist in both the log and the snapshot of a topic partition.

The `ControlRecordType` is **TBD** and will be updated when the code is committed to Kafka.

```
{
  "type": "data",
  "name": "AddVoterRecord",
  "validVersions": "0",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "Version", "type": "int16", "versions": "0+",
      "about": "The version of the add voter record"},
    { "name": "VoterId", "type": "int32", "versions": "0+", "entityType": "brokerId",
      "about": "The ID of the voter getting added to the topic partition"},
    { "name": "VoterUuid", "type": "uuid", "versions": "0+",
      "about": "The replica generated UUID of the replica getting added as a voter to the topic partition" }
  ]
}
```

Handling

KRaft replicas will read all of the control records in the snapshot and the log irrespective of the commit state and HWM. When a replica encounters an AddVoterRecord it will add the replica ID and UUID to its voter set. If the replica getting added is itself then it will allow the transition to candidate when the fetch timer expires. The fetch timer is reset whenever it receives a successful Fetch or FetchSnapshot response.

RemoveVoterRecord

A control record for instructing the voters to remove a new voter to the topic partition. This record can exist in the log but not the snapshot of a topic partition.

The `ControlRecordType` is **TBD** and will be updated when the code is committed to Kafka.

```
{
  "type": "data",
  "name": "RemoveVoterRecord",
  "validVersions": "0",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "Version", "type": "int16", "versions": "0+",
      "about": "The version of the add voter record"},
    { "name": "VoterId", "type": "int32", "versions": "0+", "entityType": "brokerId",
      "about": "The ID of the voter getting removed from the topic partition"},
    { "name": "VoterUuid", "type": "uuid", "versions": "0+",
      "about": "The replica generated UUID of the replica getting removed as a voter from the topic partition"}
  ]
}
```

Handling

KRaft replicas will read all of the control records in the snapshot and the log irrespective of the commit state and HWM. When a replica encounters a RemoveVoterRecord it will remove the replica ID and UUID from its voter set. If the replica getting removed is the leader and is the local replica then the replica will stay leader until the RemoveVoterRecord gets committed or the epoch advances losing leadership of the latest epoch.

Quorum State

Each KRaft topic partition has a quorum state (QuorumStateData) that gets persisted in the qurorum-state.json file in the directory for the topic partition.

A new field called VotedUuid will get added to QuorumStateData . This is the UUID for the replica for which the local replica voted in the LeaderEpoch.

Remove the CurrentVoters field from QuorumStateData . This information will instead being persisted in the log and snapshot using the AddVoterRecord and RemoveVoterRecord.

```
{
  "type": "data",
  "name": "QuorumStateData",
  "validVersions": "0-1",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "ClusterId", "type": "string", "versions": "0+"},
    { "name": "LeaderId", "type": "int32", "versions": "0+", "default": "-1"},
    { "name": "LeaderEpoch", "type": "int32", "versions": "0+", "default": "-1"},
    { "name": "VotedId", "type": "int32", "versions": "0+", "default": "-1"},
    { "name": "VotedUuid", "type": "uuid", "versions": "1+", "nullableVersions": "1+", "default": "null" },
    { "name": "AppliedOffset", "type": "int64", "versions": "0+"},
    { "name": "CurrentVoters", "type": "[]Voter", "versions": "0", "nullableVersions": "0"}
  ],
  "commonStructs": [
    { "name": "Voter", "versions": "0", "fields": [
      {"name": "VoterId", "type": "int32", "versions": "0"}
    ]}
  ]
}
```

meta.properties

A new property called storage.id will get added to the meta.properties of a log directory. If the meta.properties file doesn't exist for the cluster metadata partition the Kafka node will fail to start. If the meta.properties file exist but it doesn't contain this property a new one will be generated and the meta.properties files will be atomically replaced. The kafka-storage CLI tool will be extended to generate and write the storage id when the format command is used.

RPCs

AddVoter

Request

This RPC can be sent by an administrative client to add a voter to the set of voters. This RPC can be sent to a broker or controller. The broker will forward the request to the controller.

```
{
  "apiKey": "TBD",
  "type": "request",
  "listeners": ["controller", "broker"],
  "name": "AddVoterRequest",
  "validVersions": "0",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "ClusterId", "type": "string", "versions": "0+" }
    { "name": "Topics", "type": "[]TopicData", "versions": "0+", "fields": [
      { "name": "TopicName", "type": "string", "versions": "0+", "entityType": "topicName",
        "about": "The name of the topic." },
      { "name": "Partitions", "type": "[]PartitionData", "versions": "0+", "fields": [
        { "name": "Index", "type": "int32", "versions": "0+",
          "about": "The partition index." },
        { "name": "VoterId", "type": "int32", "versions": "0+",
          "about": "The ID of the voter getting added to the topic partition." },
        { "name": "VoterUuid", "type": "uuid", "versions": "0+",
          "about": "The replica generated UUID of the replica getting added as a voter to the topic partition."
}
      ]}
    ]}
  ]
}
```

Response

```
{
  "apiKey": "TBD",
  "type": "response",
  "name": "AddVoterResponse",
  "validVersions": "0",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "ErrorCode", "type": "int16", "versions": "0+",
      "about": "The top level error code." }
    { "name": "Topics", "type": "[]TopicData", "versions": "0+", "fields": [
      { "name": "TopicName", "type": "string", "versions": "0+", "entityType": "topicName",
        "about": "The name of the topic." },
      { "name": "Partitions", "type": "[]PartitionData", "versions": "0+", "fields": [
        { "name": "Index", "type": "int32", "versions": "0+",
          "about": "The partition index." },
        { "name": "ErrorCode", "type": "int16", "versions": "0+",
          "about": "The error code, or 0 if there was no fetch error." },
        { "name": "CurrentLeader", "type": "LeaderIdAndEpoch", "versions": "0+", "taggedVersions": "0+", "tag":
0, "fields": [
          { "name": "LeaderId", "type": "int32", "versions": "0+", "default": "-1", "entityType" : "brokerId",
            "about": "The ID of the current leader or -1 if the leader is unknown." },
          { "name": "LeaderEpoch", "type": "int32", "versions": "0+", "default": "-1",
            "about": "The latest known leader epoch." }
        ]}
      ]}
    ]}
  ]
}
```

Handling

When the leader receives a AddVoter request it will do the following:

1. Wait for the fetch offset of the replica (ID, UUID) to catch up to the log end offset of the leader.
2. Wait for until there are no uncommitted add or remove voter records.
3. Wait for the LeaderChangeMessage control record from the current epoch to get committed.
4. Append the AddVoterRecord to the log.
5. The KRaft internal listener will read this record from the log and add the voter to the voter set.
6. Wait for the AddVoterRecord to commit using the majority of new configuration.
7. Send the AddVoter response to the client.

In 1., the leader needs to wait for the replica to catch up because when the AddVoterRecord is appended to the log the set of voter changes. If the added voter is far behind then it can take some time for it to reach the HWM. During this time the leader cannot commit data and the quorum will be unavailable from the perspective of the state machine. We mitigate this by waiting for the new replica to catch up before adding it to the set of voters.

In 3., the leader needs to wait for LeaderChangeMessage of the current leader epoch to get committed to address this known issue.

In 5., the new replica will be part of the quorum so the leader will start sending BeginQuorumEpoch requests to this replica. It is possible that the new replica has not yet replicated and applied this AddVoterRecord so it doesn't know that it is a voter for this topic partition. The new replica will fail this RPC until it discovers that it is in the voter set. The leader will continue to retry until the RPC succeeds.

The replica will return the following errors:

1. NOT_LEADER_FOR_PARTITION - when the request is sent to a replica that is not the leader.
2. VOTER_ALREADY_ADDED - when the request contains a replic ID and UUID that is already in the committed voter set.
3. INVALID_REQUEST - when the request contains a replica ID and UUID that is not an observer.
4. INVALID_REQUEST - when the request contains a replica ID that is not enumerated in the controller.quorum.voters configuration.

The broker will handle this request by forwarding the request to the active controller.

RemoveVoter

Request

This RPC can be sent by an administrative client to remove a voter from the set of voters. This RPC can be sent to a broker or controller. The broker will forward the request to the controller.

```
{
  "apiKey": "TBD",
  "type": "request",
  "listeners": ["controller", "broker"],
  "name": "RemoveVoterRequest",
  "validVersions": "0",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "Topics", "type": "[]TopicData", "versions": "0+", "fields": [
      { "name": "TopicName", "type": "string", "versions": "0+", "entityType": "topicName",
        "about": "The name of the topic." },
      { "name": "Partitions", "type": "[]PartitionData", "versions": "0+", "fields": [
        { "name": "Index", "type": "int32", "versions": "0+",
          "about": "The partition index." },
        { "name": "VoterId", "type": "int32", "versions": "0+",
          "about": "The ID of the voter getting removed from the topic partition." },
        { "name": "VoterUuid", "type": "uuid", "versions": "0+",
          "about": "The replica generated UUID of the replica getting removed as a voter from the topic
partition." },
      ]}
    ]}
  ]
}
```

Response

```
{
  "apiKey": "TBD",
  "type": "response",
  "name": "RemoveVoterResponse",
  "validVersions": "0",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "ErrorCode", "type": "int16", "versions": "0+",
      "about": "The top level error code." }
    { "name": "Topics", "type": "[]TopicData", "versions": "0+", "fields": [
      { "name": "TopicName", "type": "string", "versions": "0+", "entityType": "topicName",
        "about": "The name of the topic." },
      { "name": "Partitions", "type": "[]PartitionData", "versions": "0+", "fields": [
        { "name": "Index", "type": "int32", "versions": "0+",
          "about": "The partition index." },
        { "name": "ErrorCode", "type": "int16", "versions": "0+",
          "about": "The error code, or 0 if there was no fetch error." },
        { "name": "CurrentLeader", "type": "LeaderIdAndEpoch", "versions": "0+", "taggedVersions": "0+", "tag":
0, "fields": [
          { "name": "LeaderId", "type": "int32", "versions": "0+", "default": "-1", "entityType" : "brokerId",
            "about": "The ID of the current leader or -1 if the leader is unknown." },
          { "name": "LeaderEpoch", "type": "int32", "versions": "0+", "default": "-1",
            "about": "The latest known leader epoch." }
        ]}
      ]}
    ]}
  ]
}
```

Handling

When the leader receives a RemoveVoter request it will do the following:

1. Wait for until there are no uncommitted add or remove voter records.
2. Wait for the LeaderChangeMessage control record from the current epoch to get committed.
3. Append the RemoveVoterRecord to the log.
4. The KRaft internal listener will read this record from the log and remove the voter from the voter set.
5. Wait for the RemoveVoterRecord to commit using the majority of new configuration.
6. Send the RemoveVoter response to the client.
7. Resign by sending EndQuorumEpoch RPCs if the removed replica is the leader.

In 3. and 5. it is possible for the RemoveVoterRecord would remove the current leader from the voter set. In this case the leader needs to allow Fetch and FetchSnapshot requests from replicas. The leader should not count itself when determining the majority and determining if records have been committed.

The replica will return the following errors:

1. NOT_LEADER_FOR_PARTITION - when the request is sent to a replica that is not the leader.
2. VOTER_ALREADY_REMOVED - when the request contains a replic ID and UUID that is already not in the committed voter set.
3. INVALID_REQUEST - when the request contains a replica ID and UUID that is not a voter.
4. INVALID_REQUEST - when the request contains a replica ID that would cause the replica ID to get removed from the voter set. In other words RemoveVoter can be used to remove replica UUIDs not replica IDs.

The broker will handle this request by forwarding the request to the active controller.

Fetch

The response versions will include version 14 and the fields will remain the same.

Request

Version 14 adds the field ReplicaUuid to the FetchPartition. This field is populated with the replica generated UUID. If the ReplicaUuid and the ReplicaId fields are populated, the topic partition leader can assume that the replica supports becoming a voter.

```
{
  "apiKey": 1,
  "type": "request",
  "listeners": ["zkBroker", "broker", "controller"],
  "name": "FetchRequest",
  "validVersions": "0-14",
  "flexibleVersions": "12+",
  "fields": [
    { "name": "ClusterId", "type": "string", "versions": "12+", "nullableVersions": "12+", "default": "null",
"taggedVersions": "12+", "tag": 0, "ignorable": true,
      "about": "The clusterId if known. This is used to validate metadata fetches prior to broker
registration." },
    { "name": "ReplicaId", "type": "int32", "versions": "0+", "entityType": "brokerId",
      "about": "The replica ID of the follower, of -1 if this request is from a consumer." },
    ...
    { "name": "Topics", "type": "[]FetchTopic", "versions": "0+",
      "about": "The topics to fetch.", "fields": [
      { "name": "Topic", "type": "string", "versions": "0-12", "entityType": "topicName", "ignorable": true,
        "about": "The name of the topic to fetch." },
      { "name": "TopicId", "type": "uuid", "versions": "13+", "ignorable": true,
        "about": "The unique topic ID"},
      { "name": "Partitions", "type": "[]FetchPartition", "versions": "0+",
        "about": "The partitions to fetch.", "fields": [
        { "name": "Partition", "type": "int32", "versions": "0+",
          "about": "The partition index." },
        { "name": "ReplicaUuid", "type": "uuid", "versions": "14+", "nullableVersions": "14+", "default":
"null",
          "about": "The replica generated UUID. null otherwise." },
        ...
      ]}
    ]}
  ]
}
```

Handling

Replica that support becoming voters will send both the replica ID and UUID in the Fetch request. The leader will assume that replicas that report both fields are voters or are able to become voters.

There are a few changes to the leader request handling described in KIP-595. The leaders will track the fetch offset for the replica tuple (ID, UUID). This means that replica are unique identified by their ID and UUID. So their state will be tracking using the ID and UUID.

When removing the leader from the voter set, it will remain the leader for that epoch until the RemoveVoterRecord gets committed. This means that the leader needs to allow replicas (voters and observers) to fetch from the leader even if it is not part of the voter set. This also means that if the leader is not part of the voter set it should not include itself when computing the committed offset (also known as the high-watermark).

FetchSnapshot

The response versions will include version 1 and the fields will remain the same.

Request

Version 1 adds the field ReplicaUuid to PartitionSnapshot. If the ReplicaUuid and the ReplicaId fields are populated, the topic partition leader can assume that the replica supports becoming a voter.

```
{
  "apiKey": 59,
  "type": "request",
  "listeners": ["controller"],
  "name": "FetchSnapshotRequest",
  "validVersions": "0-1",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "ClusterId", "type": "string", "versions": "0+", "nullableVersions": "0+", "default": "null",
"taggedVersions": "0+", "tag": 0,
      "about": "The clusterId if known, this is used to validate metadata fetches prior to broker
   registration" },
    { "name": "ReplicaId", "type": "int32", "versions": "0+", "default": "-1", "entityType": "brokerId",
      "about": "The replica ID of the follower" },
    { "name": "MaxBytes", "type": "int32", "versions": "0+", "default": "0x7fffffff",
      "about": "The maximum bytes to fetch from all of the snapshots" },
    { "name": "Topics", "type": "[]TopicSnapshot", "versions": "0+",
      "about": "The topics to fetch", "fields": [
      { "name": "Name", "type": "string", "versions": "0+", "entityType": "topicName",
        "about": "The name of the topic to fetch" },
      { "name": "Partitions", "type": "[]PartitionSnapshot", "versions": "0+",
        "about": "The partitions to fetch", "fields": [
        { "name": "Partition", "type": "int32", "versions": "0+",
          "about": "The partition index" },
        { "name": "ReplicaUuid", "type": "uuid", "versions": "1+", "default": "null",
           "about": "The replica UUID of the follower" },
        { "name": "CurrentLeaderEpoch", "type": "int32", "versions": "0+",
          "about": "The current leader epoch of the partition, -1 for unknown leader epoch" },
        { "name": "SnapshotId", "type": "SnapshotId", "versions": "0+",
          "about": "The snapshot endOffset and epoch to fetch", "fields": [
          { "name": "EndOffset", "type": "int64", "versions": "0+" },
          { "name": "Epoch", "type": "int32", "versions": "0+" }
        ]},
        { "name": "Position", "type": "int64", "versions": "0+",
          "about": "The byte position within the snapshot to start fetching from" }
      ]}
    ]}
  ]
}
```

Handling

Similar to Fetch, replica that support becoming voters will send both the replica ID and UUID in the FetchSnapshot request. The leader will assume that replicas that report both fields are voters or are able to become voters.

There are a few changes to the leader request handling described in KIP-630. The leaders will track the fetch offset for the replica tuple (ID, UUID). This means that replica are unique identified by their ID and UUID. So their state will be tracking using the ID and UUID.

When removing the leader from the voter set, it will remain the leader for that epoch until the RemoveVoterRecord gets committed. This means that the leader needs to allow replicas (voters and observers) to fetch snapshots from the leader even if it is not part of the voter set.

Vote

Request

Changes:

1. Candidate Id was moved out of the topic partition maps
2. Candidate Uuid was added to the PartitionData
3. VoterId was added to the top level
4. VoterUuId was added to PartitionData

```
{
  "apiKey": 52,
  "type": "request",
  "listeners": ["controller"],
  "name": "VoteRequest",
  "validVersions": "0-1",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "ClusterId", "type": "string", "versions": "0+", "nullableVersions": "0+", "default": "null" },
    { "name": "CandidateId", "type": "int32", "versions": "1+", "entityType": "brokerId",
      "about": "The ID of the voter sending the request" },
    { "name": "VoterId", "type": "int32", "versions": "1+", "entityType": "brokerId",
      "about": "The ID of the replica receiving the request to vote." },
    { "name": "Topics", "type": "[]TopicData", "versions": "0+", "fields": [
      { "name": "TopicName", "type": "string", "versions": "0+", "entityType": "topicName",
        "about": "The topic name." },
      { "name": "Partitions", "type": "[]PartitionData",
        "versions": "0+", "fields": [
        { "name": "PartitionIndex", "type": "int32", "versions": "0+",
          "about": "The partition index." },
        { "name": "CandidateEpoch", "type": "int32", "versions": "0+",
          "about": "The bumped epoch of the candidate sending the request"},
        { "name": "CandidateId", "type": "int32", "versions": "0", "entityType": "brokerId",
          "about": "The ID of the voter sending the request"},
        { "name": "CandidateUuid", "type": "uuid", "versions": "1+",
          "about": "The candidate generated UUID, null otherwise" },
        { "name": "VoterUuid", "type": "uuid", "versions": "1+", "nullableVersions": "1+", "default": "null" }
          "about": "The replica generated UUID of the replica receiving the request to vote, null if not known
by the candidate." },
        { "name": "LastOffsetEpoch", "type": "int32", "versions": "0+",
          "about": "The epoch of the last record written to the metadata log"},
        { "name": "LastOffset", "type": "int64", "versions": "0+",
          "about": "The offset of the last record written to the metadata log"}
      ]}
    ]}
  ]
}
```

Response

Version 1 adds the field VoterUuid to PartitionData.

```
{
  "apiKey": 52,
  "type": "response",
  "name": "VoteResponse",
  "validVersions": "0-1",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "ErrorCode", "type": "int16", "versions": "0+",
      "about": "The top level error code."},
    { "name": "Topics", "type": "[]TopicData",
      "versions": "0+", "fields": [
      { "name": "TopicName", "type": "string", "versions": "0+", "entityType": "topicName",
        "about": "The topic name." },
      { "name": "Partitions", "type": "[]PartitionData",
        "versions": "0+", "fields": [
        { "name": "PartitionIndex", "type": "int32", "versions": "0+",
          "about": "The partition index." },
        { "name": "ErrorCode", "type": "int16", "versions": "0+"},
        { "name": "LeaderId", "type": "int32", "versions": "0+", "entityType": "brokerId",
          "about": "The ID of the current leader or -1 if the leader is unknown."},
        { "name": "LeaderEpoch", "type": "int32", "versions": "0+",
          "about": "The latest known leader epoch"},
        { "name": "VoteGranted", "type": "bool", "versions": "0+",
          "about": "True if the vote was granted and false otherwise"},
        { "name": "VoterUuid", "type": "bool", "versions": "1+",
          "about": "The replica generated uuid for the replica casting a vote." }
      ]}
    ]}
  ]
}
```

Handling

The candidate replica will only send vote requests to replicas it considers voters. The handling of the Vote request will change from that described in KIP-595 in that all replicas are allowed to cast a vote even if they are not voters for the quorum. The voter will persist in the quorum state both the candidate ID and UUID so that it only votes for at most one candidate for a given epoch.

The replica will return the following errors:

1. INVALID_REQUEST - when the voter ID and UUID doesn't match the local ID and UUID. Note that the UUID is optional will only be checked if provided.

BeginQuorumEpoch

The version of the response is increase and the fields remain unchanged.

Request

1. LeaderId was moved out of the topic partition maps
2. VoterId was added to the top level
3. VoterUuId was added to PartitionData
4. Allow tagged fields for version greater than or equal to 1.

```
{
  "apiKey": 53,
  "type": "request",
  "listeners": ["controller"],
  "name": "BeginQuorumEpochRequest",
  "validVersions": "0-1",
  "flexibleVersions": "1+",
  "fields": [
    { "name": "ClusterId", "type": "string", "versions": "0+",
      "nullableVersions": "0+", "default": "null"},
    { "name": "LeaderId", "type": "int32", "versions": "1+", "entityType": "brokerId",
      "about": "The ID of the newly elected leader"},
    { "name": "VoterId", "type": "int32", "versions": "1+", "entityType": "brokerId",
      "about": "The voter ID of the receiving replica." },
    { "name": "Topics", "type": "[]TopicData", "versions": "0+", "fields": [
      { "name": "TopicName", "type": "string", "versions": "0+", "entityType": "topicName",
        "about": "The topic name." },
      { "name": "Partitions", "type": "[]PartitionData", "versions": "0+", "fields": [
        { "name": "PartitionIndex", "type": "int32", "versions": "0+",
          "about": "The partition index." },         ,
        { "name": "VoterUuid", "type": "uuid", "versions": "1+", "nullableVersions": "1+", "default": "null" }
          "about": "The replica generated UUID of the replica receiving the request to vote, null if not known
by the leader." },
        { "name": "LeaderId", "type": "int32", "versions": "0", "entityType": "brokerId",
          "about": "The ID of the newly elected leader"},
        { "name": "LeaderEpoch", "type": "int32", "versions": "0+",
          "about": "The epoch of the newly elected leader"}
      ]}
    ]}
  ]
}
```

Handling

This request will be handle as described in KIP-595 with the following additional errors:

1. INVALID_REQUEST - when the voter ID and UUID doesn't match. Note that the UUID is optional will only be checked if provided.
2. NOT_VOTER - when the the replica is not a voter in the topic partition. This error should be retry by the leader of the topic partition.

EndQuorumEpoch

The version of the response is increase and the fields remain unchanged.

Request

1. LeaderId was moved out of the topic partition maps
2. VoterId was added to the request
3. VoterUuid was added to the Partitions
4. ReplicaUuid was added to PreferredSuccessors
5. Allow tagged fields for versions greater than or equal to 1.

```
{
  "apiKey": 54,
  "type": "request",
  "listeners": ["controller"],
  "name": "EndQuorumEpochRequest",
  "validVersions": "0-1",
  "flexibleVersions": "1+",
  "fields": [
    { "name": "ClusterId", "type": "string", "versions": "0+",
      "nullableVersions": "0+", "default": "null"},
    { "name": "LeaderId", "type": "int32", "versions": "1+", "entityType": "brokerId",
      "about": "The current leader ID that is resigning." },
    { "name": "VoterId", "type": "int32", "versions": "1+", "entityType": "brokerId",
      "about": "The voter ID of the receiving replica." },
    { "name": "Topics", "type": "[]TopicData", "versions": "0+", "fields": [
      { "name": "TopicName", "type": "string", "versions": "0+", "entityType": "topicName",
        "about": "The topic name." },
      { "name": "Partitions", "type": "[]PartitionData", "versions": "0+", "fields": [
        { "name": "PartitionIndex", "type": "int32", "versions": "0+",
          "about": "The partition index." },         ,
        { "name": "VoterUuid", "type": "uuid", "versions": "1+", "nullableVersions": "1+", "default": "null" }
          "about": "The replica generated UUID of the replica receiving the request to vote, null if not known
by the leader." },
        { "name": "LeaderId", "type": "int32", "versions": "0", "entityType": "brokerId",
          "about": "The current leader ID that is resigning"},
        { "name": "LeaderEpoch", "type": "int32", "versions": "0+",
          "about": "The current epoch"},
        { "name": "PreferredSuccessors", "type": "[]ReplicaInfo", "versions": "0+",
          "about": "A sorted list of preferred successors to start the election", "fields": [
          { "name": "ReplicaId", "type": "int32", "versions": "0+", "entityType": "brokerId" },
          { "name": "ReplicaUuid", "type": "uuid", "versions": "1+" }
        ]}
      ]}
    ]}
  ]
}
```

Handling

This request will be handle as described in KIP-595 with the following additional errors:

1. INVALID_REQUEST - when the voter ID and UUID doesn't match.
2. NOT_VOTER - when the the replica is not a voter in the topic partition. This error could be retry by the leader of the topic partition.

DescribeQuorum

The version of the request is increase and the fields remain unchanged.

Response

1. Add ReplicaUuid to ReplicaState

```
{
  "apiKey": 55,
  "type": "response",
  "name": "DescribeQuorumResponse",
  "validVersions": "0-2",
  "flexibleVersions": "0+",
  "fields": [
    { "name": "ErrorCode", "type": "int16", "versions": "0+",
      "about": "The top level error code."},
    { "name": "Topics", "type": "[]TopicData",
      "versions": "0+", "fields": [
      { "name": "TopicName", "type": "string", "versions": "0+", "entityType": "topicName",
        "about": "The topic name." },
      { "name": "Partitions", "type": "[]PartitionData",
        "versions": "0+", "fields": [
        { "name": "PartitionIndex", "type": "int32", "versions": "0+",
          "about": "The partition index." },
        { "name": "ErrorCode", "type": "int16", "versions": "0+"},
        { "name": "LeaderId", "type": "int32", "versions": "0+", "entityType": "brokerId",
          "about": "The ID of the current leader or -1 if the leader is unknown."},
        { "name": "LeaderEpoch", "type": "int32", "versions": "0+",
          "about": "The latest known leader epoch"},
        { "name": "HighWatermark", "type": "int64", "versions": "0+"},
        { "name": "CurrentVoters", "type": "[]ReplicaState", "versions": "0+" },
        { "name": "Observers", "type": "[]ReplicaState", "versions": "0+" }
      ]}
    ]}
  ],
  "commonStructs": [
    { "name": "ReplicaState", "versions": "0+", "fields": [
      { "name": "ReplicaId", "type": "int32", "versions": "0+", "entityType": "brokerId" },
      { "name": "ReplicaUuid", "type": "uuid", "versions": "2+" },
      { "name": "LogEndOffset", "type": "int64", "versions": "0+",
        "about": "The last known log end offset of the follower or -1 if it is unknown"},
      { "name": "LastFetchTimestamp", "type": "int64", "versions": "1+", "ignorable": true, "default": -1,
        "about": "The last known leader wall clock time time when a follower fetched from the leader. This is
reported as -1 both for the current leader or if it is unknown for a voter" },
      { "name": "LastCaughtUpTimestamp", "type": "int64", "versions": "1+", "ignorable": true, "default": -1,
        "about": "The leader wall clock append time of the offset for which the follower made the most recent
fetch request. This is reported as the current time for the leader and -1 if unknown for a voter" }
    ]}
  ]
}
```

Handling

The handling of the request is the same as that described in KIP-595 with just the additional fields. Clients handling the response can assume that if a ReplicaState include both the ID and the UUID that replica can become a voter if is enumerated in the Observers field.

Admin Client

The Java Admin client will be extended to support the new field in the DescribeQuorum response and the new AddVoter and RemoveVoter RPCs.

Monitoring

NAME	TAGS	TYPE	NOTE
number-of-voters	type=raft-metrics	gauge	number of voters for the cluster metadata topic partition.
number-of-possible-voters	type=raft-metrics	guage	number of observer that could be promoted to voters.
pending-add-voter	type=raft-metrics	guage	1 if there is a pending add voter operation, 0 otherwise.
pending-remove-voter	type=raft-metrics	guage	1 if there is a pending remove voter operation, 0 otherwise.

Command Line Interface

kafka-metadata-shell.sh

A future KIP will describe how the kafka-metadata-shell tool will be extended to be able to read and display KRaft control records from the quorum, snapshot and log.

kafka-storage.sh

The kafka-storage CLI tool will generate a storage UUID and meta.properties file for every data log directory and the metadata log directory.

kafka-metadata-quorum.sh

The kafka-metadata-quorum tool described in KIP-595 and KIP-836 will be improved to support these additional commands:

--describe

This command will print both the ReplicaId and ReplicaUuid for CurrentVoters. A new row called CouldBeVoters will be added which print the Replica ID and UUID of any replica that could be added to the voter set. E.g.

```
> bin/kafka-metadata-quorum.sh --describe
ClusterId:              SomeClusterId
LeaderId:               0
LeaderEpoch:            15
HighWatermark:          234130
MaxFollowerLag:         34
MaxFollowerLagTimeMs:   15
CurrentVoters:          [{"id": 0, "uuid": "UUID1"}, {"id": 1, "uuid": "UUID2"}, {"id": 2, "uuid": "UUID2"}]
CouldBeVoters:          [{"id": 3, "uuid": "UUID3"}]
```

--describe replication

This command will print on additional column for the replica uuid after the replica id. E.g.

```
> bin/kafka-metadata-quorum.sh --describe replication
ReplicaId   ReplicaUuid   LogEndOffset   ...
0           uuid1         234134         ...
...
```

--add-voter

This command is used to add new voters to the topic partition. The flags --replicaId and --replicaUuid must be specified.

--remove-voter

This command is used to remove voters from the topic partition. The flags --replicaId and --replicaUuid must be specified.

Compatibility, Deprecation, and Migration Plan

The features in this KIP will be supported if the ApiVersions of all of the voters and observers is greater than the versions described here. If the leader has a replica UUID for all of the voters then this KIP is supported by all of the voters.

Test Plan

This KIP will be tested using unittest, integration tests, system test, simulation tests and TLA+ specification.

Rejected Alternatives

KIP-642: Dynamic quorum reassignment. KIP-642 describe how to perform dynamic reassignment will multiple voters added and removed. KIP-642 doesn't support disk failures and it would be more difficult to implement compared to this KIP-856. In a future KIP we can describe how we can add administrative operations that support the addition and removal of multiple voters.

References

1. Ongaro, Diego, and John Ousterhout. "In search of an understandable consensus algorithm." *2014 {USENIX} Annual Technical Conference ({USENIX}{ATC} 14)*. 2014.
2. Ongaro, Diego. *Consensus: Bridging theory and practice*. Diss. Stanford University, 2014.
3. Bug in single-server membership changes
4. KIP-595: A Raft Protocol for the Metadata Quorum
5. KIP-630: Kafka Raft Snapshot
6. KIP-631: The Quorum-based Kafka Controller