Bringing Kafka Without ZooKeeper Into Production

Colin McCabe
Principal Engineer
Confluent

About Me

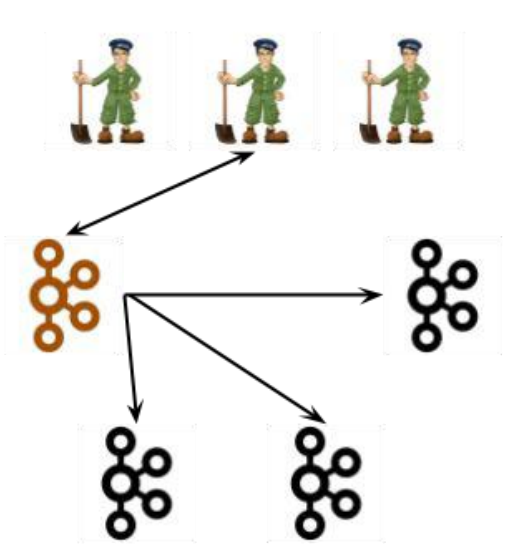
I work on Apache Kafka at Confluent.

Kafka Committer & PMC Member

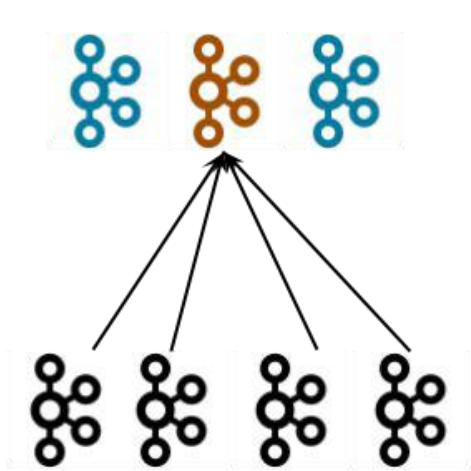
Table of Contents

KRaft Architecture

Removing ZooKeeper



Current
ZK-Based
Architecture



New
KRaft-Based
Architecture

How Kafka Uses ZooKeeper

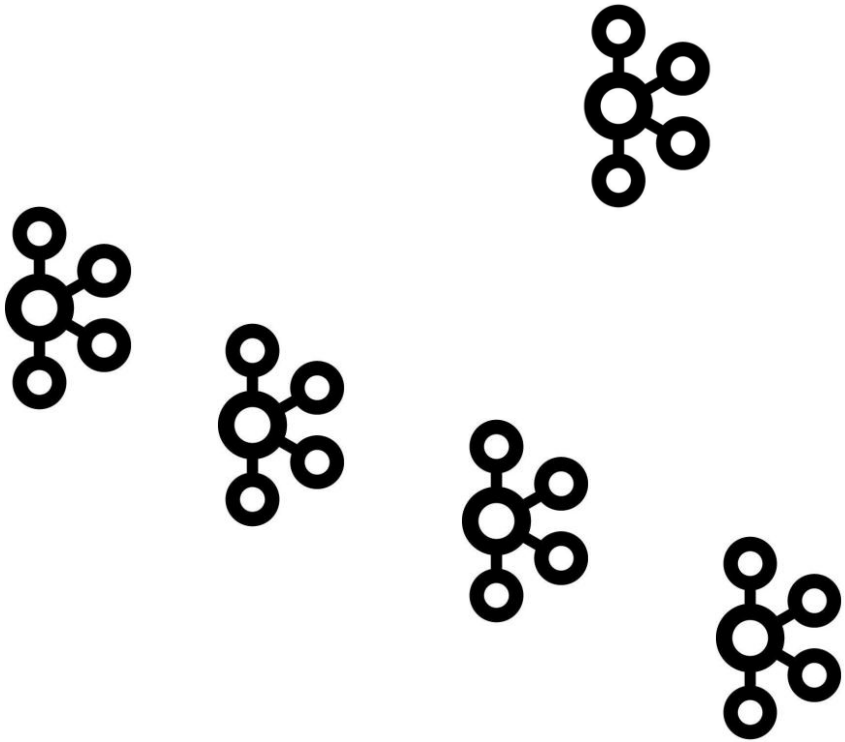
Stores most persistent metadata in ZooKeeper

- Topics
- Partitions
- Configurations
- Quotas
- ACLs

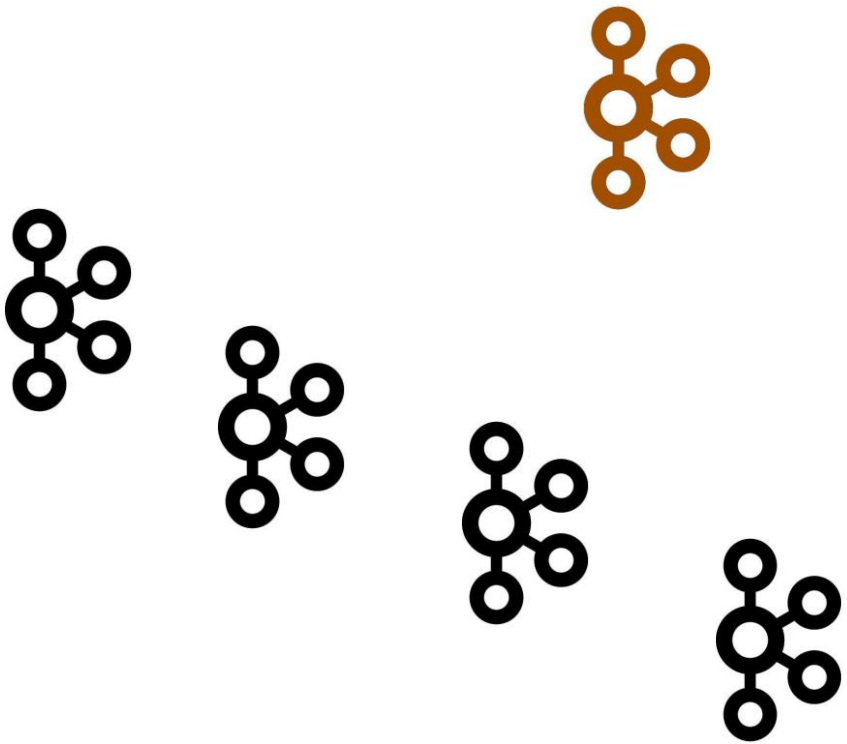
Also uses ZooKeeper for coordinating cluster membership.

/brokers/ids/0
/brokers/ids/1
/brokers/topics/foo
/brokers/topics/foo/partitions/0/state
/brokers/topics/foo/partitions/1/state
/brokers/topics/foo/partitions/2/state
…

Controller Startup with ZK

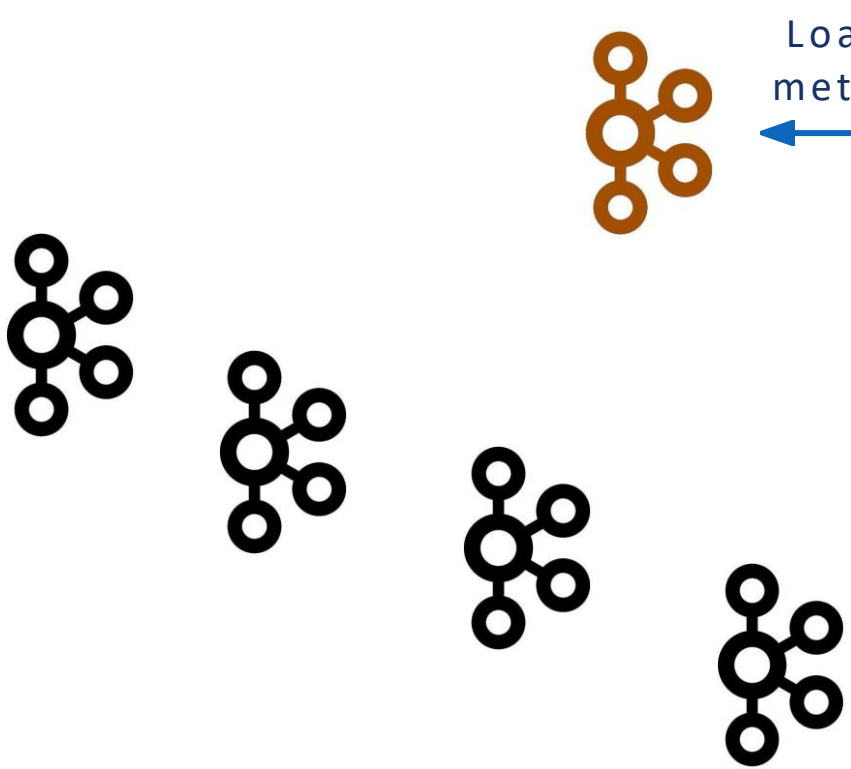


Controller Startup with ZK

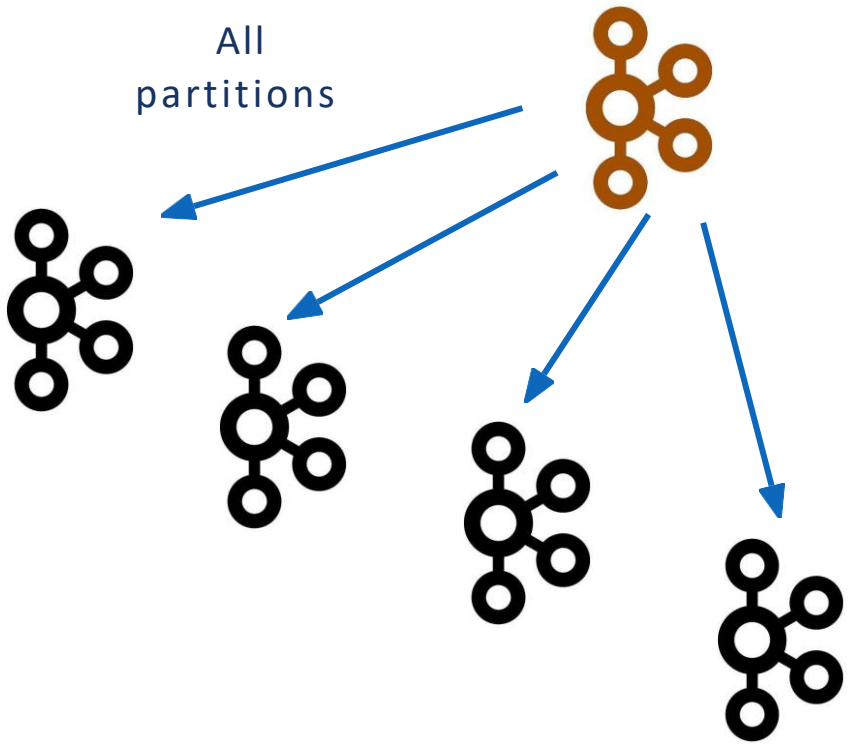


- One broker wins the election in ZooKeeper to be the controller

Controller Startup with ZK



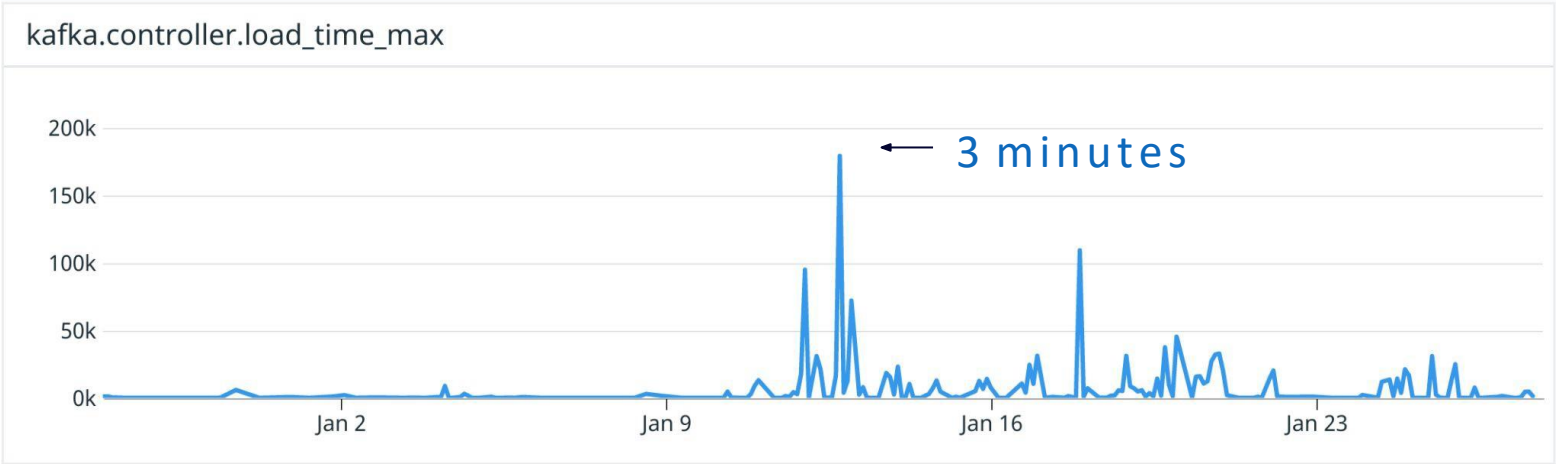
Controller Startup with ZK



- UpdateMetadataRequest
- LeaderAndIsrRequest

Problems with ZK-based Controller Startup

- Have to load **all** metadata synchronously on startup:
  - O(num_partitions), O(num_brokers)
  - Controller is unavailable during this time
    - Cold start: cluster unavailable.
    - Controller restart: admin ops and ISR changes unavailable
- Have to send **all** metadata to **all** brokers on startup



How KRaft Replaces ZooKeeper

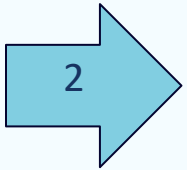
- Instead of ZooKeeper, we have an internal topic named **__cluster_metadata**
  - Replicated with KRaft
  - Single partition
  - The leader is the active controller
- KRaft: Raft for Kafka
  - The Raft protocol implemented in Kafka
    - Records committed by a majority of nodes
  - Self-managed quorum
    - Doesn't rely on an external system for leader election

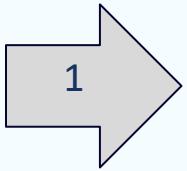
Metadata Records

- Binary records containing metadata
  - KRPC format
  - Auto-generated from protocol schemas
  - Can also be translated into JSON for human readability
- Two ways to evolve format
  - New record versions
  - Tagged fields
- Some records are deltas that apply changes to existing state

```
{
  "type": "REGISTER_BROKER_RECORD",
  "version": 0,
  "data": {
    "brokerId": 1,
    "incarnationId": "P3UFsWoNR-erL9PK98YLsA",
    "brokerEpoch": 0,
    "endPoints": [
      {
        "name": "PLAINTEXT",
        "host": "localhost",
        "port": 9092,
        "securityProtocol": 0
      }
    ],
    "features": [],
    "rack": null
  }
}
```

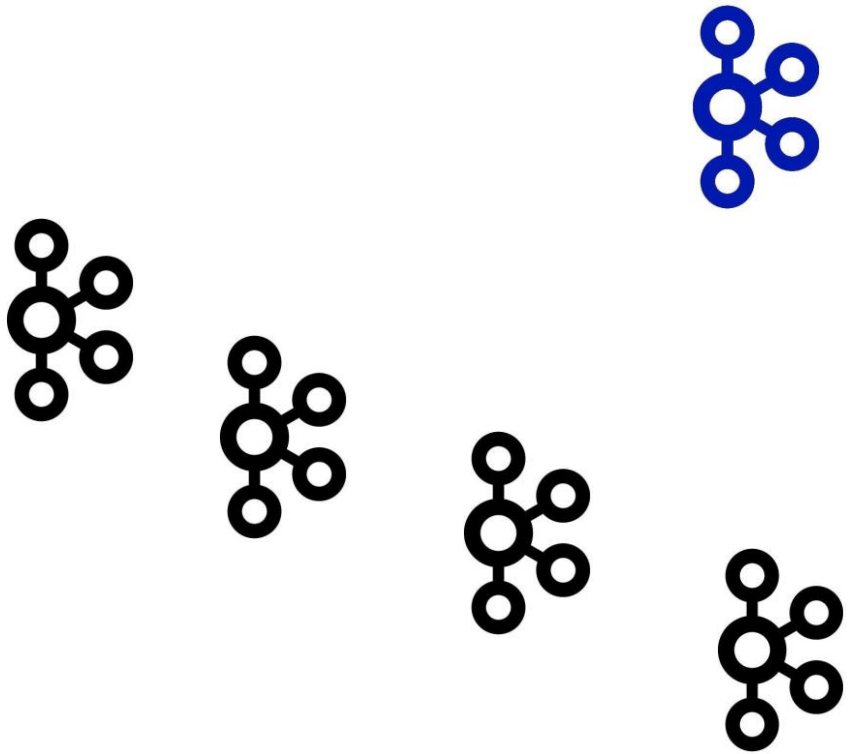
Metadata as An Ordered Log





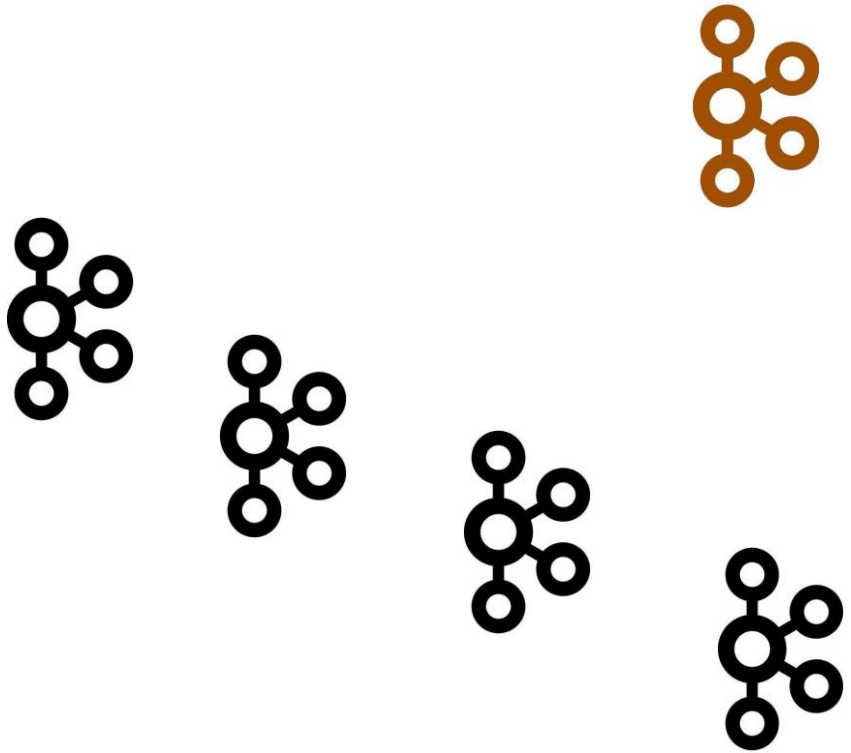
10434	TopicRecord(name=foo, id=rtkInsMkQPiEBj6uz67rrQ)
10435	PartitionRecord(id=rtkInsMkQPiEBj6uz67rrQ, index=0, …)
10436	PartitionRecord(id=rtkInsMkQPiEBj6uz67rrQ, index=1, …)
10437	PartitionRecord(id=rtkInsMkQPiEBj6uz67rrQ, index=2, …)
10438	ConfigRecord(name=num.io.threads, value=...)
10439	RegisterBrokerRecord(id=4, endpoints=…, …)

Controller Startup with KRaft



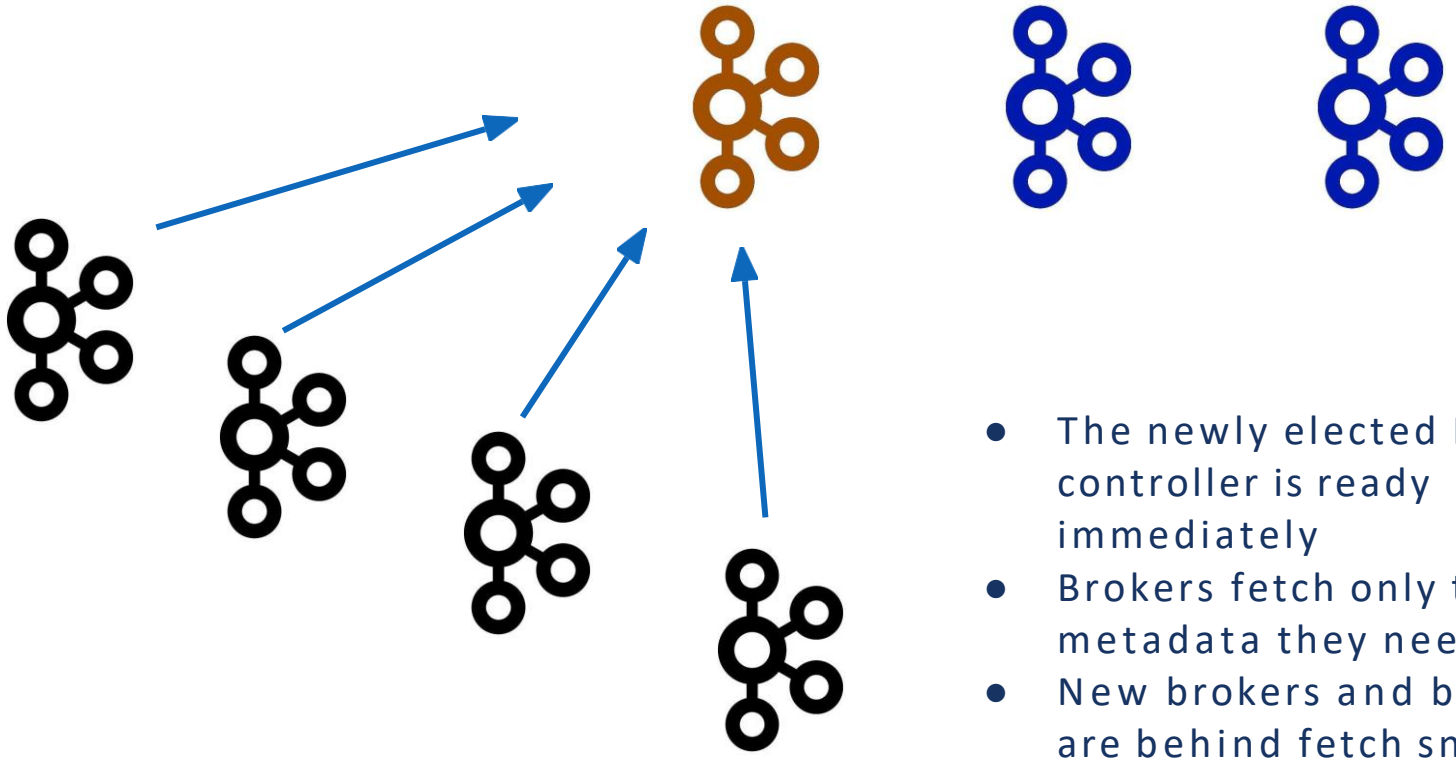
- The blue nodes are designated KRaft controllers

Controller Startup with KRaft



- The KRaft controllers will elect a single leader
- The leader will have all previously committed records

Controller Startup with KRaft



- The newly elected KRaft controller is ready immediately
- Brokers fetch only the metadata they need
- New brokers and brokers that are behind fetch snapshots

Controller Failover

ZK Mode

- Win controller election
- Load all topics and partitions
- Send LeaderAndIsr + UpdateMetadata to all brokers

KRaft Mode

- Win KRaft election
- Start handling requests from brokers

Rolling Nodes

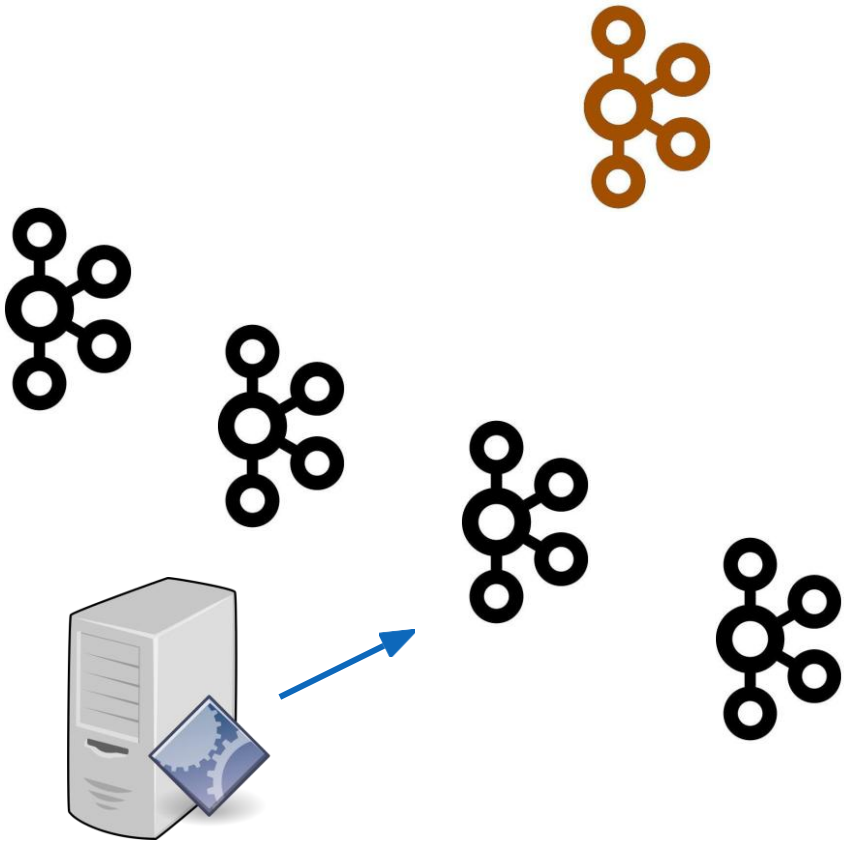
ZK Mode

Restarted node begins with no metadata. Must wait for full metadata update RPCs from controller.

KRaft Mode

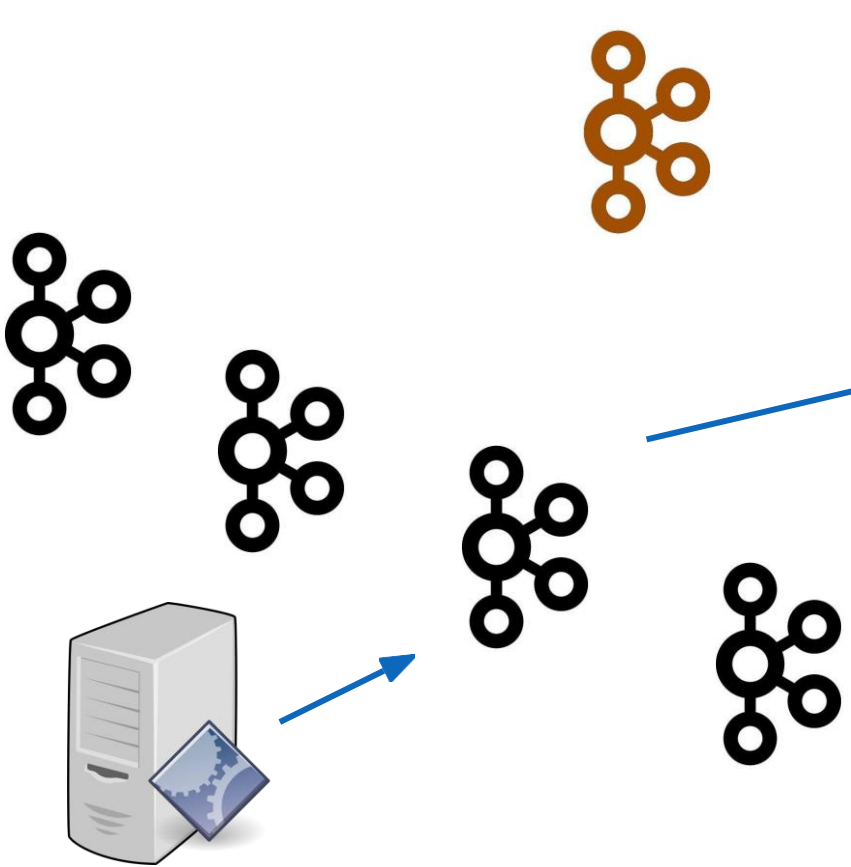
Restarted node consults its local metadata cache, transfers only what it doesn't have, based on metadata offset.

Calling an Admin API with ZK



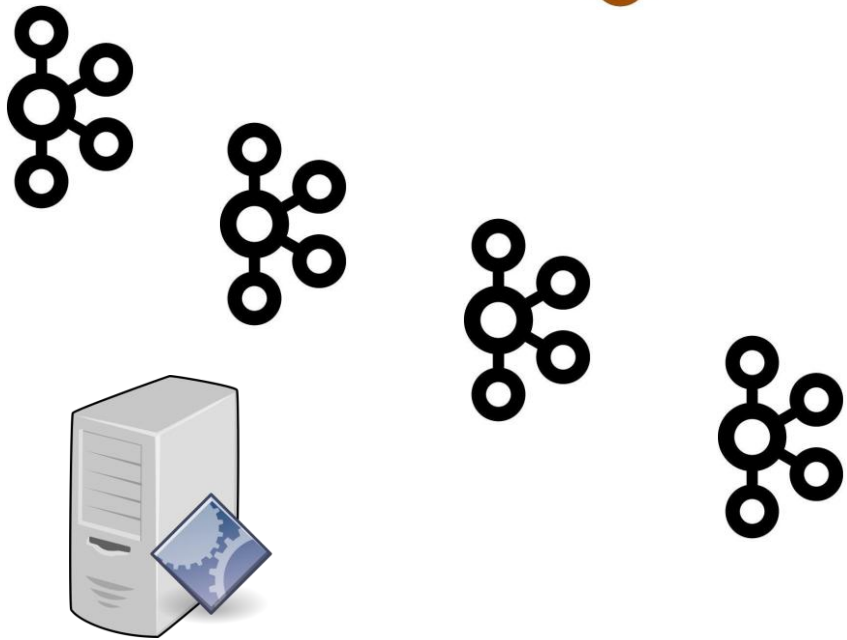
- External client connects to a random broker

Calling an Admin API with ZK



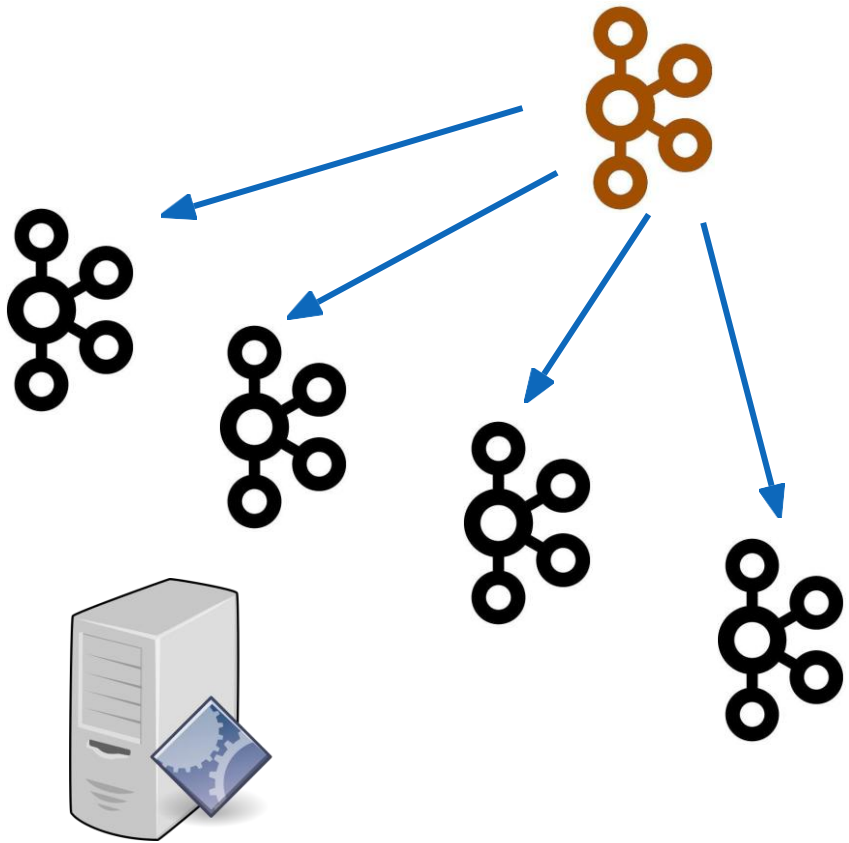
- Broker alters ZooKeeper
  - Example: create znodes for new topic

Calling an Admin API with ZK



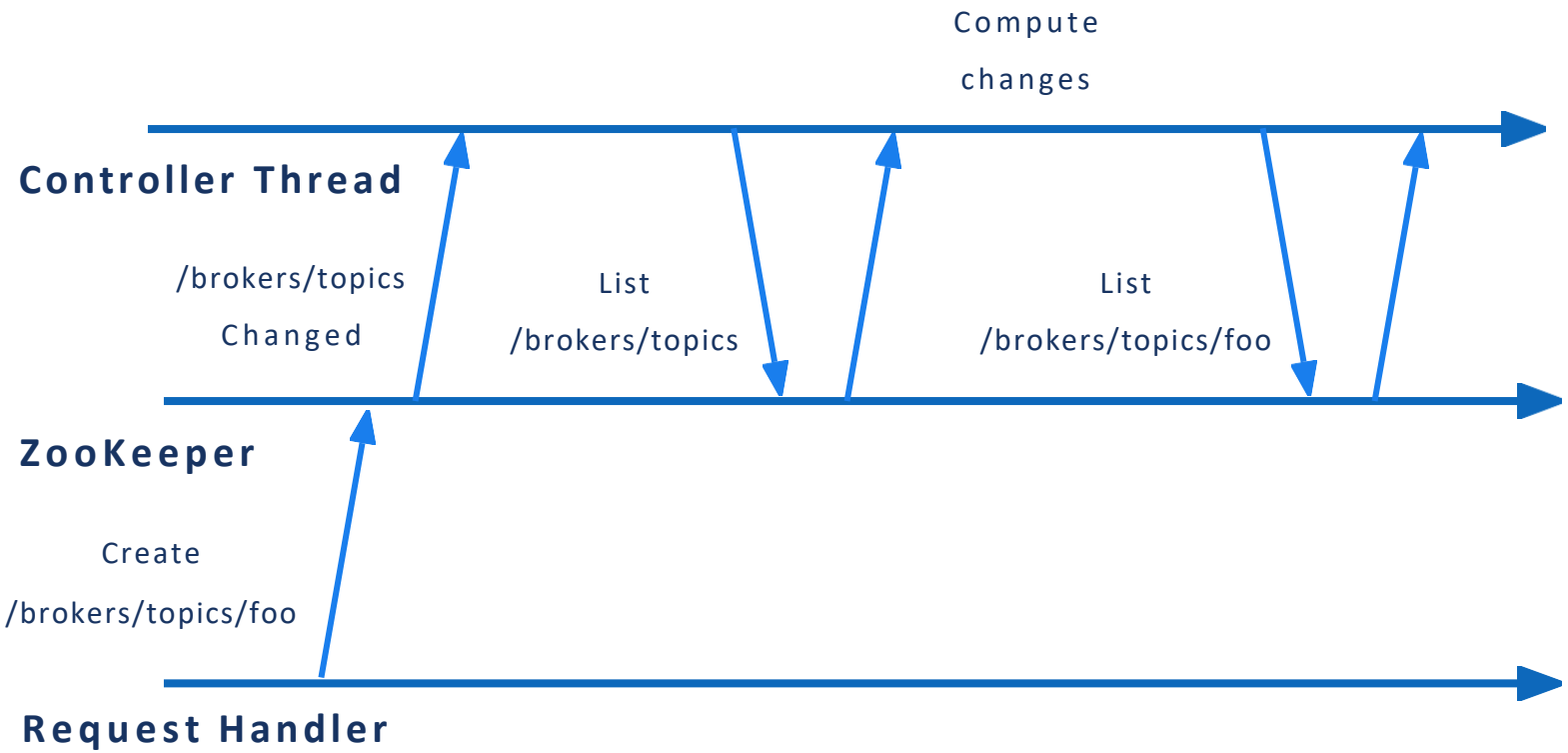
- Znode watch triggers
- Controller loads changes

Calling an Admin API with ZK



- Controller pushes out changes to brokers
- Incremental LeaderAndIsr
- Incremental UpdateMetadata

ZK-based Programming Model



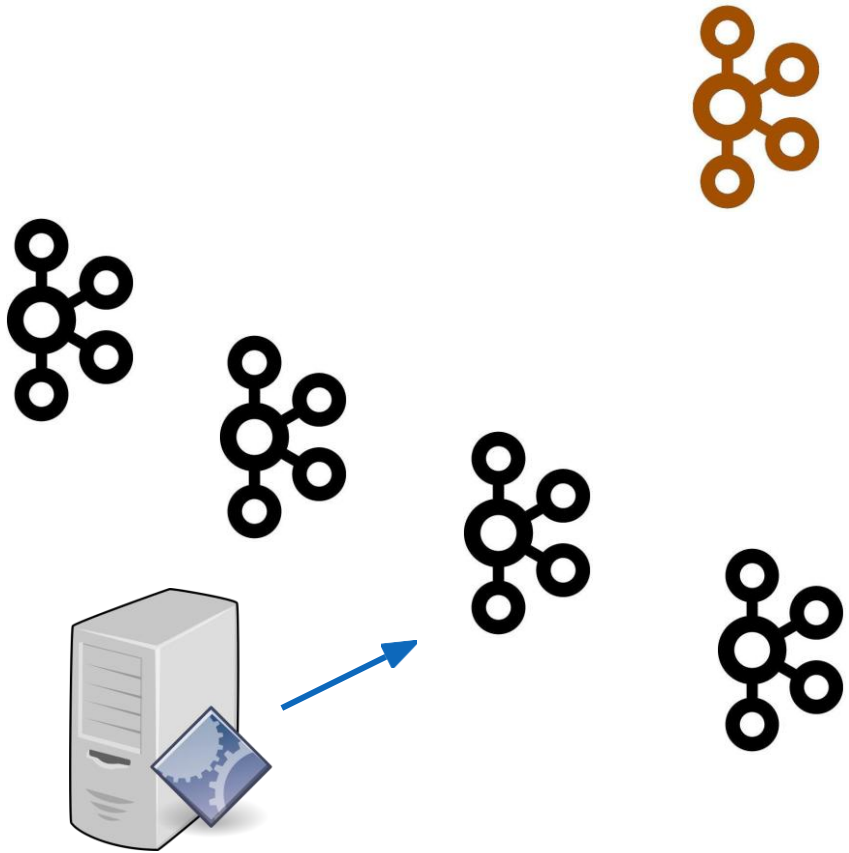
Problems

- What if the controller fails to send an update to a specific broker?
  - Metadata divergence
  - No easy way to know what we have and what we don't have

- Difficult programming model
  - Multiple writers to ZooKeeper
  - Can't assume your cache is up-to-date!

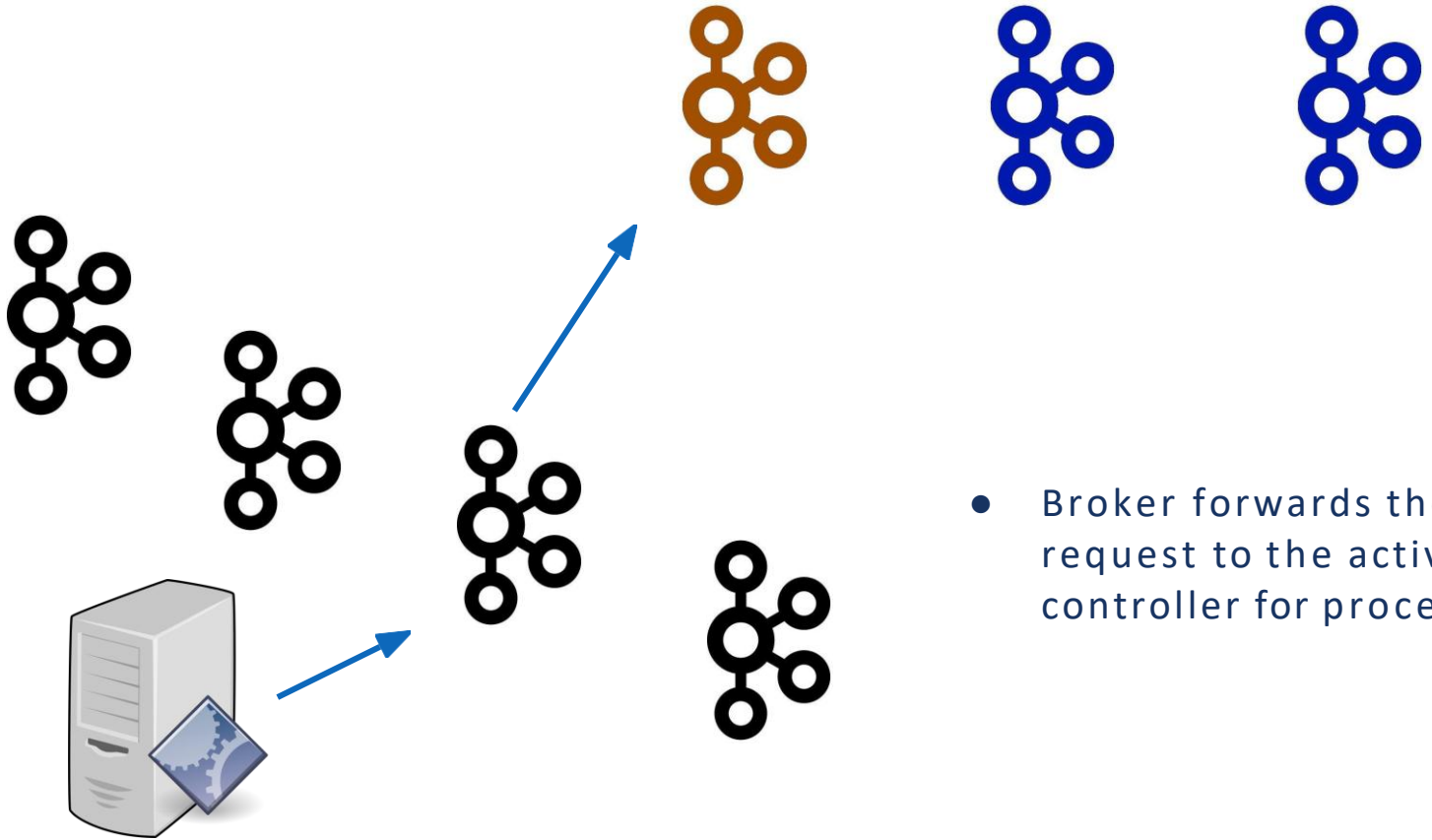
- ZK is the bottleneck for all admin operations
  - Often 1admin operation leads to many ZK operations
  - Blocking

Calling an Admin API with KRaft



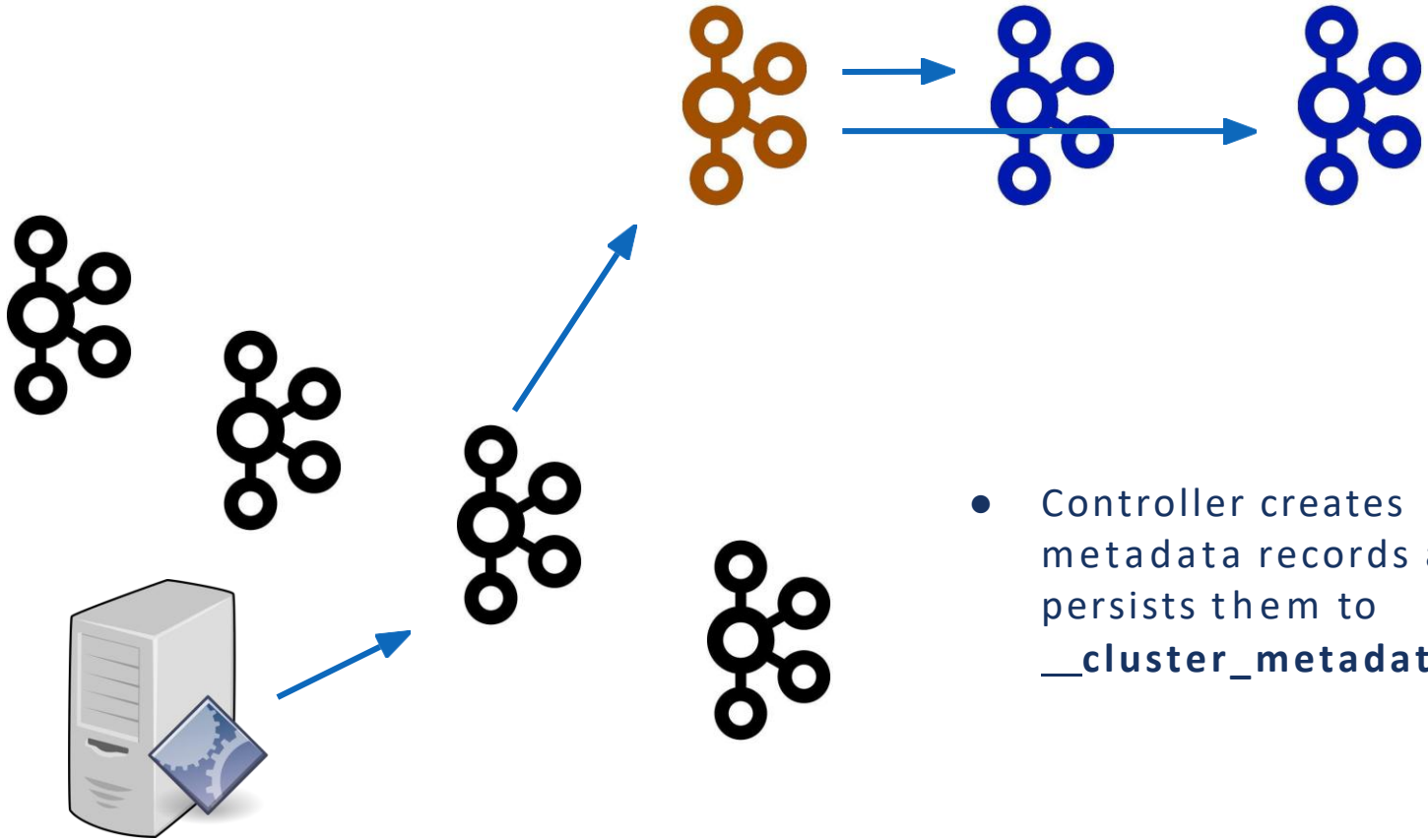
- External client connects to a random broker

Calling an Admin API with KRaft



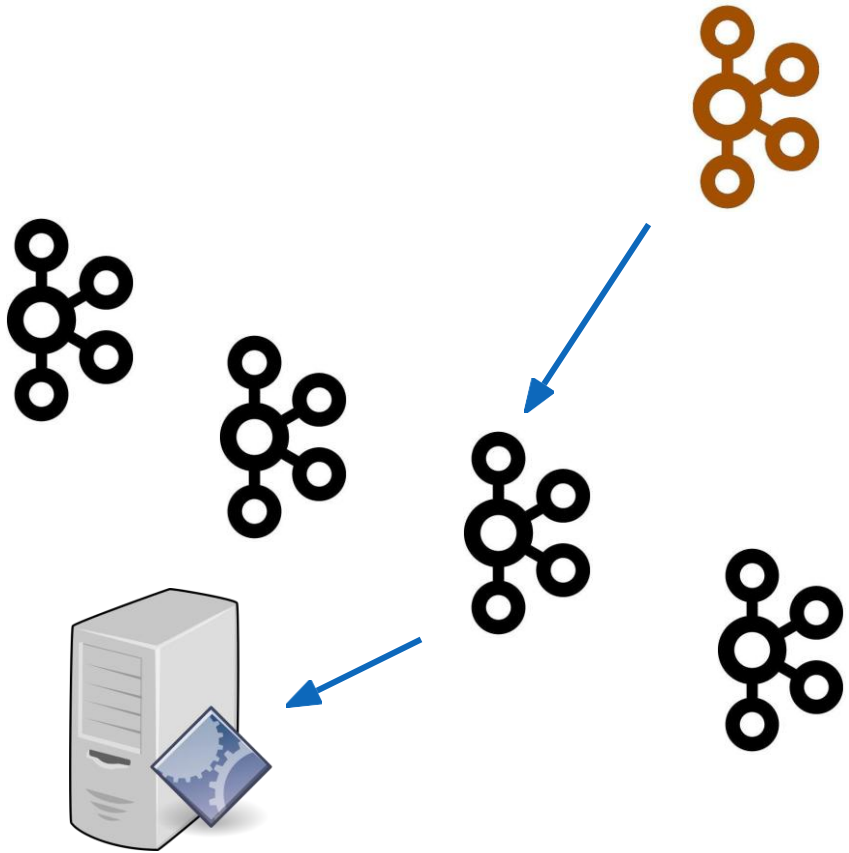
- Broker forwards the request to the active controller for processing

Calling an Admin API with KRaft



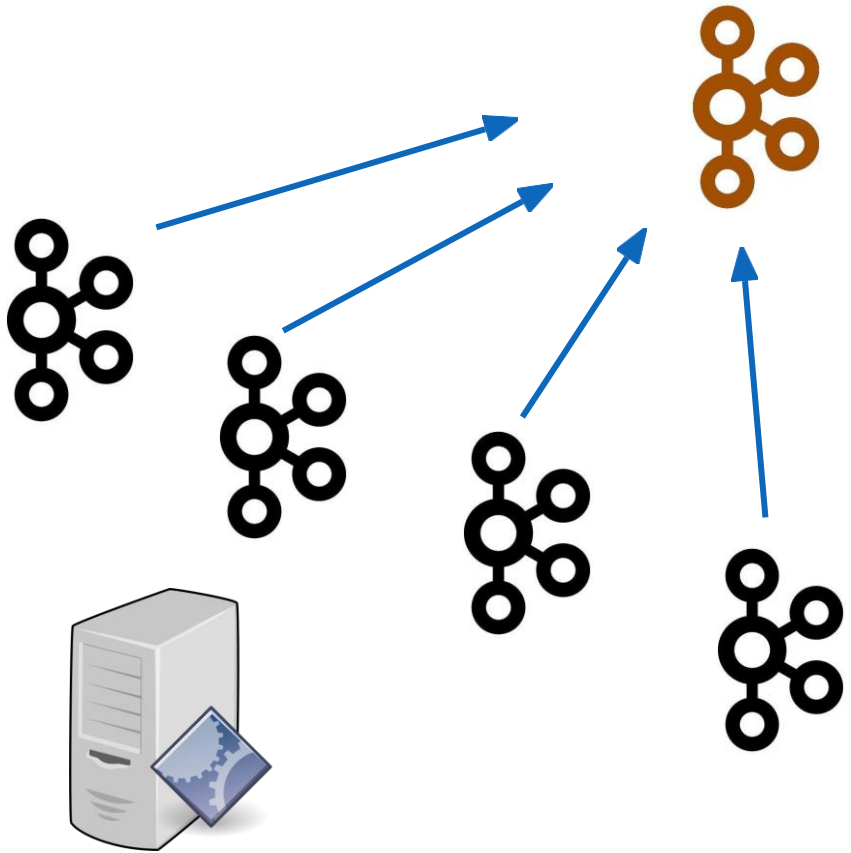
- Controller creates metadata records and persists them to **__cluster_metadata**

Calling an Admin API with KRaft



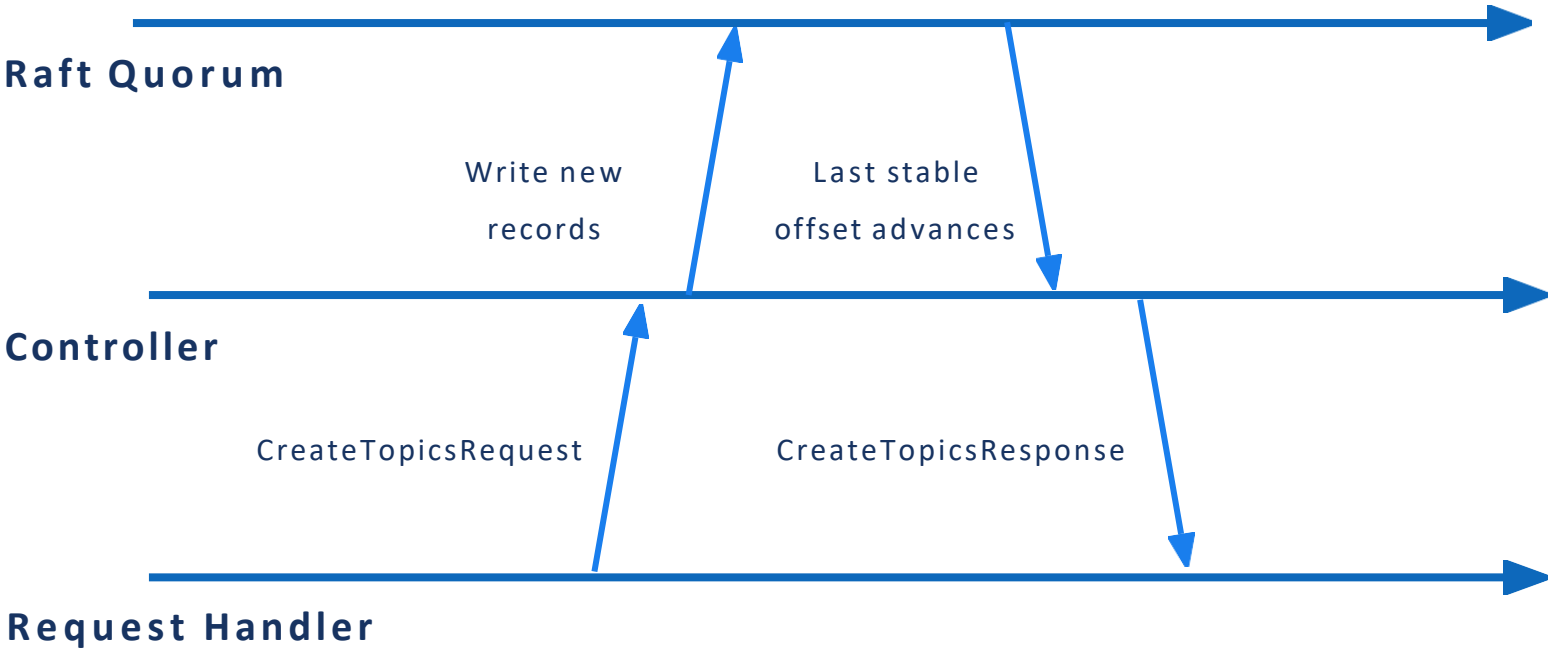
- Once the records have been committed to the metadata log, the active controller returns the result to the forwarding broker, which returns it to the external client

Calling an Admin API with KRaft

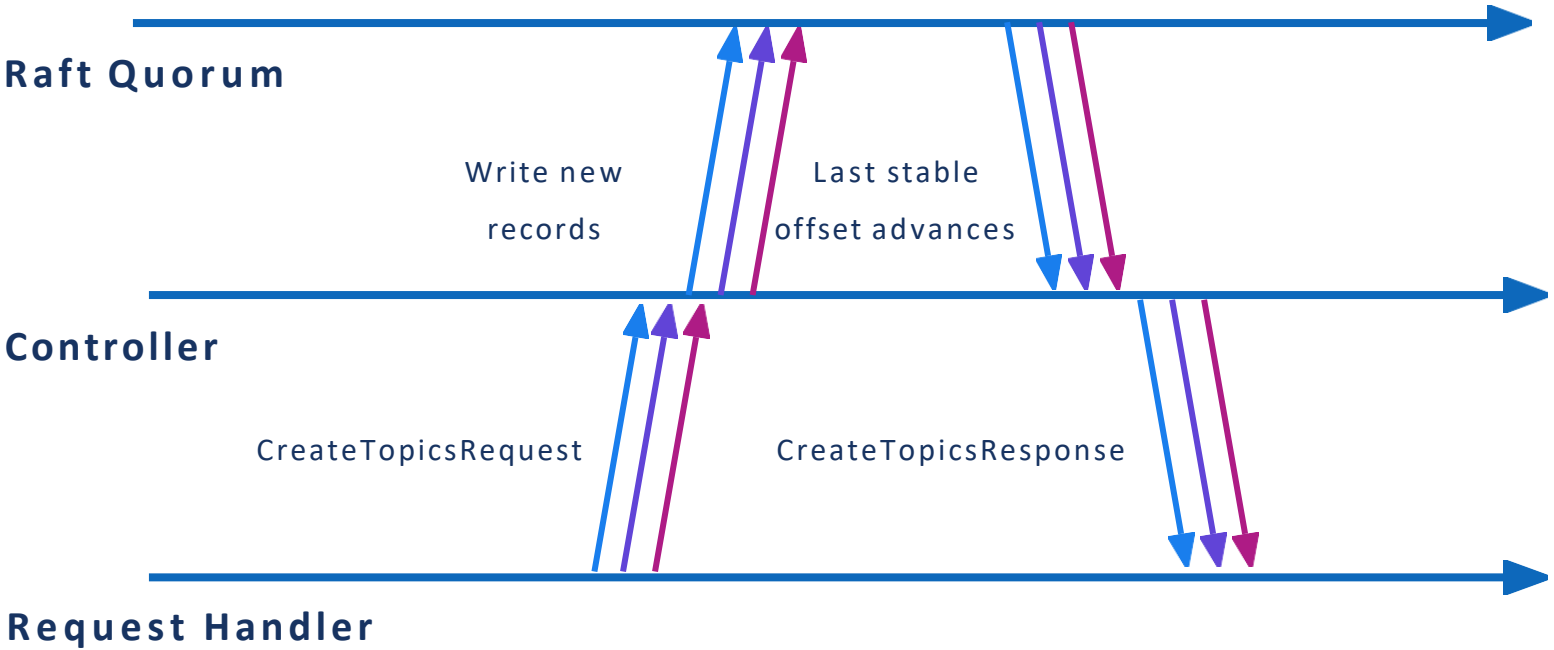


- Brokers are continuously fetching metadata from the active controller.
- They become aware of the admin changes by reading the new metadata records.

KRaft-based Programming Model



KRaft-based Programming Model: Pipelining



Admin Operations in KRaft

- Pull-based metadata propagation model can recover from RPC send failures easily
- Simpler programming model: single-writer
- Pipelining means that we can have multiple metadata operations in flight at once

Deploying KRaft

Getting KRaft

- The latest Apache (and Confluent) releases support both ZK mode and KRaft mode
  - KRaft mode is supported from the same code base – not a branch
- KRaft mode is **not** yet recommended for production
  - In Apache 2.8, KRaft was in early access
  - As of Apache 3.0, KRaft is in preview
  - Definitely use the latest version if you are trying it out – many bugs have been fixed
- ZooKeeper mode will be supported for a while
  - I'll discuss the release plans in more detail in the Roadmap section of this talk

Creating a New KRaft Cluster

- Generate a new random uuid with kafka-storage tool
- On each node, use the kafka-storage tool to format storage directories
- Start all nodes

- First two steps are new!

```
$ kafka-storage.sh random-uuid
MX1HbXq8TFymY2SYR1hGYg

$ kafka-storage.sh format \
     -c ./config/kraft/controller.properties \
     --cluster-id MX1HbXq8TFymY2SYR1hGYg
Formatting /tmp/kraft-controller-logs
```

The Role of Formatting

- In ZK mode, brokers auto-format storage directories if they are blank on startup.
- In KRaft mode, we must format each broker and controller before starting the nodes.
- Why not auto-format?
  - Avoid admin mistakes or filesystem issues that could lose data
    - If a volume is not mounted, the directory may appear empty
    - The same reasons why databases and filesystems require a formatting step.
  - We need a way to **bootstrap** security-related dynamic configurations that the quorum and the brokers need to function
    - SCRAM configurations
    - Dynamic SSL configurations
    - Metadata version

Bootstrapping

ZK Mode

- Start up ZooKeeper cluster
- Set SCRAM users, dynamic security configs, etc. in ZK
- Start brokers

KRaft Mode

- Generate cluster ID
- Run format tool on all nodes, specifying cluster ID, SCRAM, security configs, metadata version, etc.
- Start nodes

KRaft Controllers

- A small pool of nodes that we designate ahead of time
  - Hot standbys for the active controller
  - Single log directory for **__cluster_metadata**

- Sizing is very similar to ZooKeeper
  - Typically 3 controller nodes
  - May use more to provide more resilience
  - Just like with ZK, need to keep a **majority** of nodes alive

- Can be configured by copying over broker configuration file
  - Set **node.roles** to controller
  - Set **node.id**

Deploying KRaft Controllers

Combined Mode

Controller processes in the same JVM as broker processes

Shared IDs, shared JVMs, shared nodes

Can have single JVM Kafka cluster

Separate Mode

Controller processes on separate nodes

Different IDs, different JVMs, different nodes

Better isolation, easier rolls

Separate Mode with Kubernetes Bin-Packing

Controllers in separate k8s pods

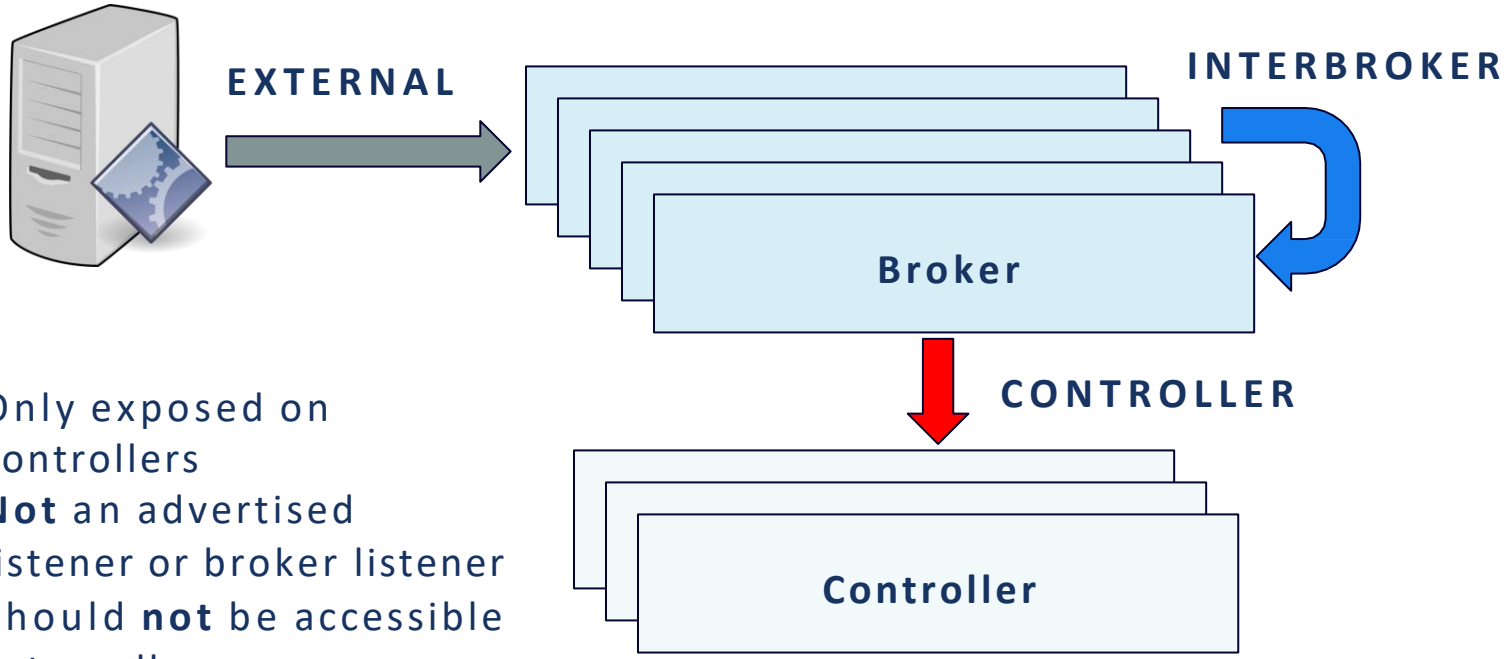
Different IDs, different JVMs, maybe shared nodes

Some Considerations for Deploying Controllers

- Isolation
  - Avoid co-location in busy clusters
  - Try to avoid situations where traffic from outside disrupts internals

- High Availability
  - Try to not to have multiple controllers in a single failure domain
  - In clusters which span 3 availability zones, put a controller in each zone

New Controller Endpoint



- Only exposed on controllers
- **Not** an advertised listener or broker listener
- Should **not** be accessible externally

Configuring KRaft Clusters

- **node.id**
  - Replaces **broker.id** on both brokers and controllers.
- **process.roles**
  - broker
  - controller
  - broker,controller
- **controller.quorum.voters**
  - 1@controller1.9093,2@controller2:9093,3@controller3:9093
  - Statically configured
- **controller.listener.names**
  - Tell brokers and other controllers how to connect to the controller
  - These are not advertised listeners and cannot appear in “advertised.listeners” on the broker
  - On the controller they appear in “listeners”

Rolling KRaft Clusters

Rolling KRaft Controllers

- Much lighter roll process
  - No cluster-destabilizing full metadata updates to send

- KRaft controllers can be rolled very quickly compared to brokers
  - They do not manage lots of data directories
  - Make sure that monitoring software can keep up

- KRaft controller software can be upgraded on a separate schedule
  - Forwards and backwards compatibility with alternate broker versions

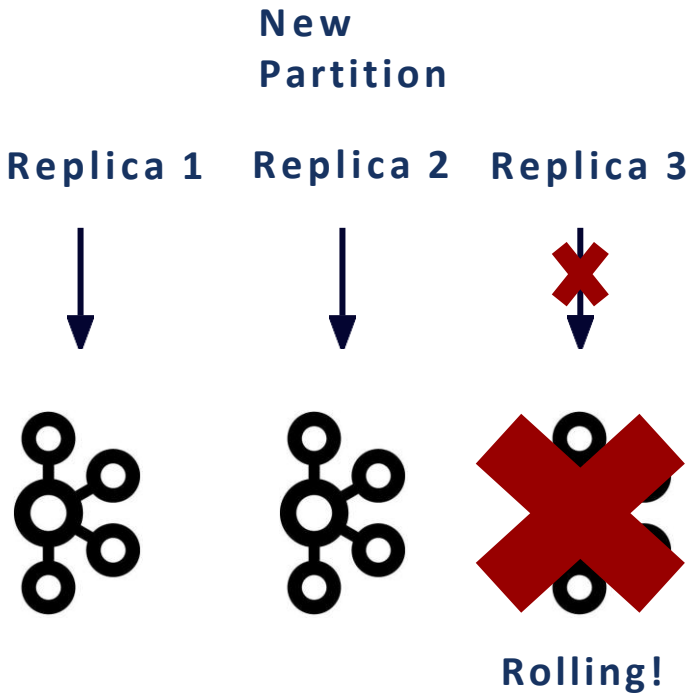
Monitoring Rolls

- **ZK Mode**
  - Under-replicated partitions
  - ActiveControllerCount
  - LeaderAndIsr request time
  - UpdateMetadata request time
  - ZooKeeperExpiresPerSec

- **KRaft Mode**
  - Under-replicated partitions
    - Shows broker health
  - ActiveControllerCount
  - MetadataOffset
  - SnapshotLag
  - SnapshotSizeBytes
  - MetadataCommitRate
  - MetadataCommitLatency
  - Current Raft state (follower, leader, candidate, observer)
    - Shows controller health
  - FencedBrokerCount
  - ActiveBrokerCount

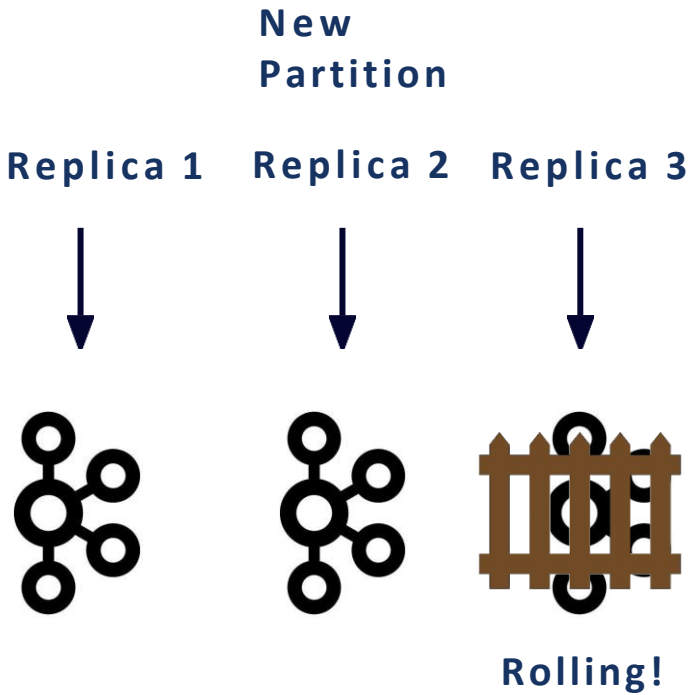
Creating New Partitions During Cluster Rolls in ZK

- In ZK mode, controller “forgets about” brokers that are temporary down during a roll
- In a 3-node cluster, if one node is rolling, we don’t have enough nodes to create a partition with 3x replication.
- Must run with 4 nodes, even if we don’t need all 4.



Creating New Partitions During Cluster Rolls in KRaft

- In KRaft mode, the controller remembers brokers that are temporary down during a roll
- In KRaft mode, those brokers enter “fenced state”
  - Can place new replicas on these nodes if needed
  - Brokers will be unfenced once they come up
- Many small clusters can now use 3 nodes rather than 4.



Handling Version Skew

- During the roll, old and new software versions must co-exist
- Older brokers can't understand newer APIs and protocols
- **inter.broker.protocol** solves this in the ZK world
  - Controls what RPC protocols brokers use when communicating with each other
  - Controls what features brokers support
- Examples
  - Whether to use AlterPartitions for ISRs
  - Whether to use topic IDs
  - Whether to use the new version of some API (ListOffsets, FetchRequest, etc.)

```
KAFKA_2_4_IV0,
KAFKA_2_4_IV1,
KAFKA_2_5_IV0,
KAFKA_2_6_IV0,
KAFKA_2_7_IV0,
KAFKA_2_7_IV1,
KAFKA_2_7_IV2,
KAFKA_2_8_IV0,
KAFKA_2_8_IV1,
KAFKA_3_0_IV0,
KAFKA_3_0_IV1,
KAFKA_3_1_IV0,
KAFKA_3_2_IV0
```

Problems with Inter Broker Protocol in ZK

- **inter.broker.protocol** version must be manually configured
  - If it is left out of the configuration file, it defaults to the latest version, which is probably not what the user wants
- Because **inter.broker.protocol** is statically configured, it can't be changed without restarting all the nodes
  - To upgrade to a new version AND get the new features requires a "double roll" in ZK mode
- No downgrade support
  - It is safe to downgrade between some pairs of versions, but not others.

Introducing metadata.version

- In KRaft mode, **inter.broker.protocol** is replaced by **metadata.version**
- Each new **inter.broker.protocol** version has a corresponding **metadata.version**
- **metadata.version** is dynamically configured
  - Does not require a roll to change
  - It is changed by invoking a controller API.
  - “Guard rails”
    - The controller will refuse to do the upgrade if some brokers have not been rolled
- Supports downgrade!

Downgrading metadata.version

- Like upgrade, downgrade can be done dynamically from the command line
- Two kinds of downgrades
  - Safe: no metadata will be lost by the downgrade
  - Unsafe: some metadata may be cleared during the downgrade
- Unsafe downgrades require the operator to provide an override flag

Troubleshooting KRaft Clusters

Troubleshooting KRaft Clusters

- In KRaft, **__cluster_metadata** replaces ZooKeeper as the store of record
  - It's often helpful to examine the metadata log to see what happened
  - All batches come with timestamps
  - Offsets are uniform across the cluster
    - A specific record will have the same offset on each controller and broker

- Several tools for looking at metadata logs
  - kafka-dump-log
  - kafka-metadata-shell

kafka-dump-log

```
$ ./bin/kafka-dump-log.sh \
--cluster-metadata-decoder \
--files /tmp/logs/__cluster_metadata-0/00000000000000000000.log

Dumping /tmp/logs/__cluster_metadata-0/00000000000000000000.log
Starting offset: 0

baseOffset: 0 lastOffset: 0 count: 1 baseSequence: -1
[...]
| offset: 0 CreateTime: 1650857270775 keySize: 4 valueSize: 19
sequence: -1 headerKeys: [] controlType: LEADER_CHANGE(2)
baseOffset: 1 lastOffset: 1 count: 1 baseSequence: -1
[...]{"type":"REGISTER_BROKER_RECORD","version":0,"data":{"...
```

Interactive Metadata Shell

- Replaces zookeeper-shell in KRaft clusters
- Data sources
  - Snapshot
  - Running controller cluster
- Reads **__cluster_metadata** log entries into memory
- Constructs a “virtual filesystem” with the cluster’s information
- Commands available: ls, pwd, cd, find, etc.

kafka-metadata-shell

```
$ ./bin/kafka-metadata-shell.sh --snapshot \
/tmp/logs/__cluster_metadata-0/00000000000000000000.log

Loading...
Starting...
[ Kafka Metadata Shell ]

>> ls
brokers  configs  local  metadataQuorum  topicIds  topics

>> cat /brokers/1/registration
RegisterBrokerRecord(brokerId=1,
incarnationId=tHo3Z8dYSuONV5hA82BVug, brokerEpoch=0,
endPoints=[BrokerEndpoint(name='PLAINTEXT', host='localhost',
port=9092, securityProtocol=0)], features=[], rack=null, fenced=true)
```

Monitoring the Quorum

- **Metrics**
  - MetadataOffset
  - SnapshotLag
  - SnapshotSizeBytes
  - MetadataCommitRate
  - MetadataCommitLatency
  - Current Raft state (follower, leader, candidate, observer)
    - Important for controller health

- **DescribeQuorum RPC**
  - Leader ID
  - Leader epoch
  - High water mark
  - Current voters
    - Controllers
  - Current observers
    - Brokers
    - Possible metadata shell instances, etc.
  - Log end offset of all followers

Upgrading from ZooKeeper Mode

Upgrading Clusters from ZK Mode

- Initially: all metadata in ZooKeeper.
- All brokers in ZK mode.

ZK mode

ZK mode

ZK mode

Upgrading Clusters from ZK Mode

KRaft controller

KRaft controller

KRaft controller

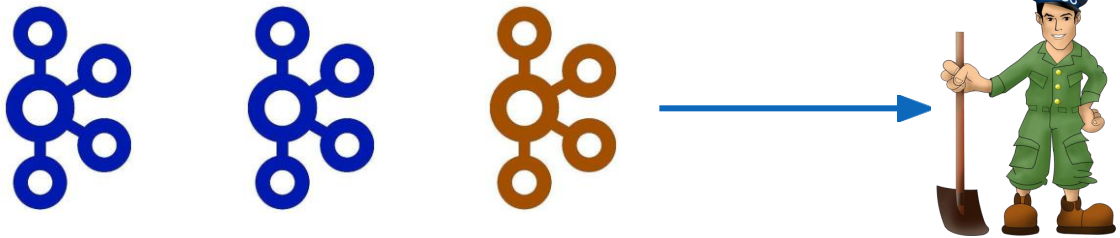
- We add a quorum of new KRaft controllers to the cluster

ZK mode

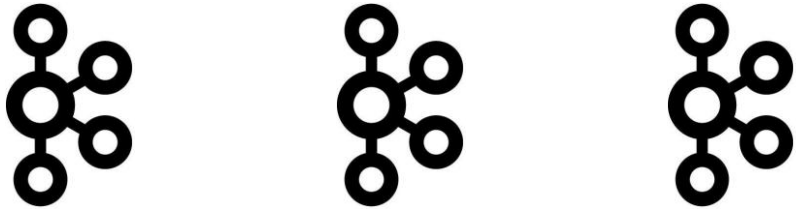
ZK mode

ZK mode

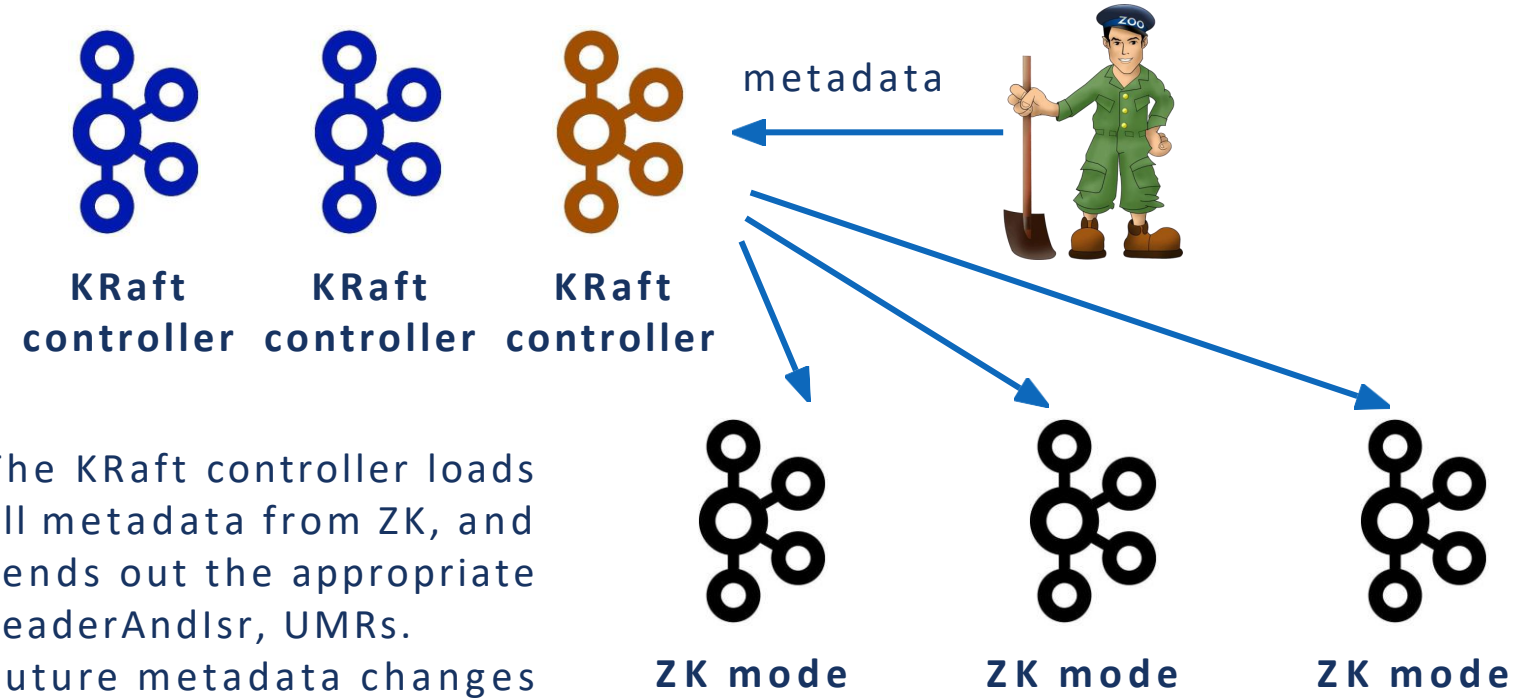
Upgrading Clusters from ZK Mode



- The KRaft controllers elect a leader from their ranks and force that leader to be the cluster leader in ZK.

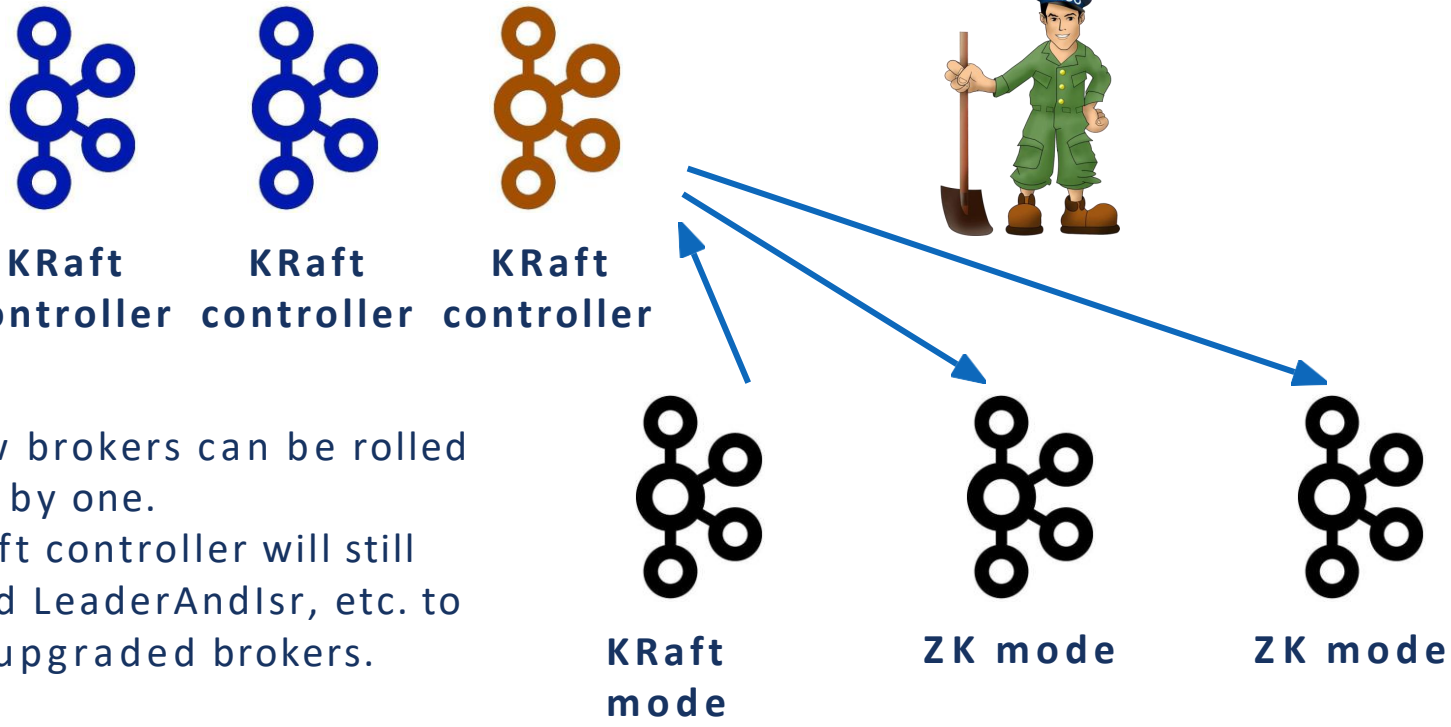


Upgrading Clusters from ZK Mode



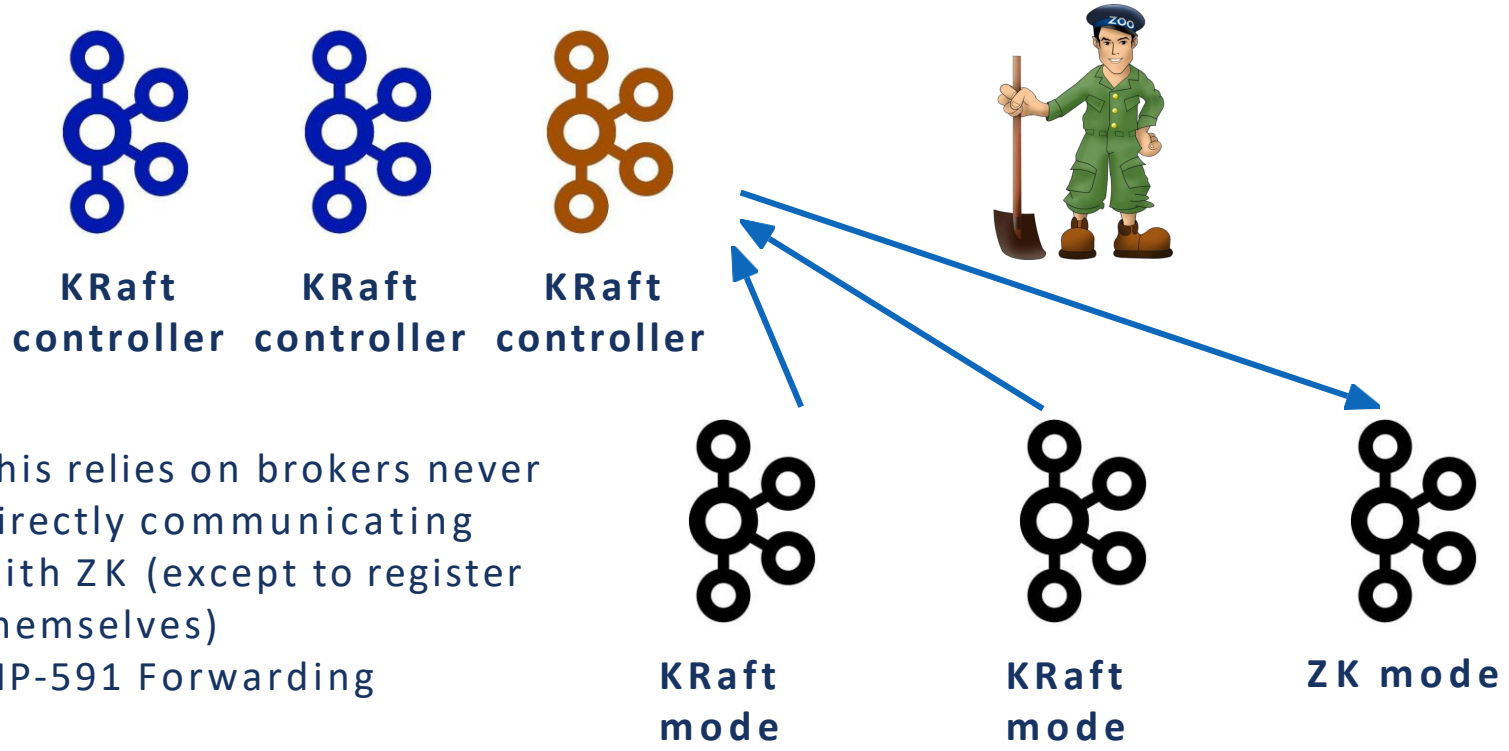
- The KRaft controller loads all metadata from ZK, and sends out the appropriate LeaderAndIsr, UMRs.
- Future metadata changes go into the KRaft quorum.

Upgrading Clusters from ZK Mode



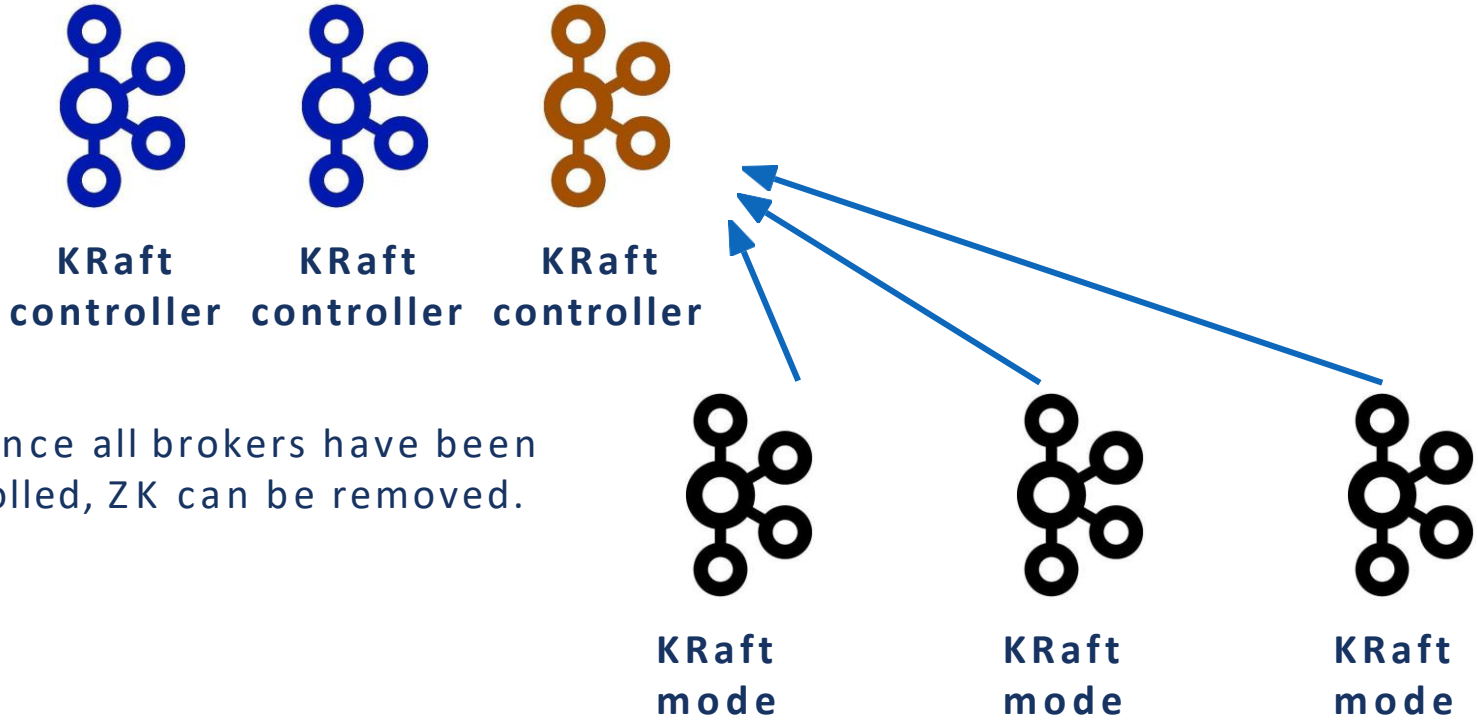
- Now brokers can be rolled one by one.
- KRaft controller will still send LeaderAndIsr, etc. to un-upgraded brokers.

Upgrading Clusters from ZK Mode



- This relies on brokers never directly communicating with ZK (except to register themselves)
- KIP-591 Forwarding

Upgrading Clusters from ZK Mode



- Once all brokers have been rolled, ZK can be removed.

Roadmap

KRaft Project Phases

1. Design
2. Initial Implementation
3. Available for testing
4. Available for production
5. Bridge release
6. ZK removed

KRaft Project Phases

1. Design
2. Initial Implementation
3. Available for testing
4. Available for production
5. Bridge release
6. ZK removed

Design Phase

- During the design phase, we had many upstream discussions about the goals of the KRaft project (called the KIP-500 project at the time) and how we would evolve the project to meet them.
- **September 2019**: vote for “KIP-500: Replace ZooKeeper with a Self-Managed Metadata Quorum” passes
- **June 2020**: vote for “KIP-591: Redirect Zookeeper Mutation Protocols to The Controller” passes (some revisions will be made later)
- **August 2020**: vote for “KIP-595: A Raft Protocol for the Quorum” passes
- **September 2020**: vote for “KIP-631: The Quorum-Based Kafka Controller” passes
- **October 2020**: vote for “KIP-630: Kafka Raft Snapshots” passes
- Several other minor KIPs filled gaps in our admin APIs which were being filled by direct ZK access

Initial Implementation Phase

- During this phase, we put the ideas from the design phase into practice.
- The implementation was pretty unstable during this time period – many big changes and fixes were happening all the time.
- There were many minor changes to remove ZK dependencies as well.
- **Late 2020**: KIP-500 work started in a branch.
- **Early 2021**: all code merged to trunk
  - New controller code
  - New self-managed Raft quorum implementation
- **April 2021**: Apache 2.8 released
  - First Apache release with KRaft
  - KRaft in Early Access
- **September 2021**: Apache 3.0 release
  - First release with KRaft snapshot support
  - Added reassignment and EOS support as well
  - Preview

Testing Phase

- This is where we are now.
- KRaft is in preview mode
  - Not recommended for production
  - Available for **testing**
- **January 2022**: Apache Kafka 3.1 released
  - Many bug fixes, new tests
  - Some new KRaft metrics
- **May 2022**: Apache Kafka 3.2 released (soon)
  - New KRaft-based authorizer

Production Phase

- Coming soon!
- During this phase, we will recommend KRaft for production use.
  - Also known as “going GA”
- There may still be some feature gaps
  - SCRAM
  - JBOD support
  - Delegation token support
- Most importantly, we will not have support for upgrading from ZK yet in this phase.
  - Therefore, in this phase, KRaft will be useful for new clusters, but not for existing ones

Bridge Release

- A bridge release is a release that supports upgrading from ZK mode to KRaft mode.
  - As described in the section on upgrading from ZK, this requires brokers to channel their admin changes through the controller, rather than mutating ZK directly.
- We will make multiple bridge releases
- We will work hard to close all the remaining feature gaps during this phase, so that there are no longer any users stuck on ZK.

Last Phase: Removing ZK Support

- Eventually, we will want to remove ZK support so that we no longer have to maintain two controller implementations
- Before we can remove ZK support, we have to deprecate it, and then make a new major release
- Timeline for full removal is TBD

Thank You!

Colin McCabe
cmccabe@apache.org