Edit Manage Stats

Brianna Blacet for Outshift By Cisco
Posted on Apr 3 • Updated on Jun 29

It's Time to Break Out of the Zoo: Running Apache Kafka® without ZooKeeper™

#kafka #zookeeper #kraft

Have you ever heard the Simon and Garfunkel song "At the Zoo?" The lyrics begin, "Someone told me it's all happening at the zoo." Well, if you're talking about Apache Kafka®, it's the exact opposite: the zookeeper has retired and gone home to watch Netflix.

In October of 2022, the Apache Software Foundation released Kafka 3.3.1—the first release that included a production-ready version of KRaft (Kafka Raft) consensus protocol. This simplified Kafka management by eliminating the need to use Apache ZooKeeper™ to manage and secure a Kafka deployment. (Note: Kafka 3.4 provides for migration from ZooKeeper to KRaft. ZooKeeper is deprecated in 3.4 and Apache plans to remove it completely in version 4.0.)

The KRaft advantage

There are a number of reasons why it made sense for Apache to deprecate ZooKeeper:

1. It's much more convenient to use one component instead of two.
2. Kafka clusters support up to 200,000 partitions. When adding Kafka brokers to or removing them from a cluster, it forces a rejiggering of leader elections. This can overload ZooKeeper and (temporarily) slow performance to a crawl. KRaft mitigates this scale problem.
3. ZooKeeper's metadata can sometimes become out of sync with Kafka's cluster metadata.
4. ZooKeeper's security lags behind Kafka's.

Enter KRaft.

KRaft is an event-based version of the Raft consensus algorithm. It uses an event log to store the state, periodically adding snapshots to save storage space. Because the state data is distributed within the metadata topic—as opposed to retrieving it from a separate tool—it dramatically improves the worst-case recovery time, decreasing the window of unavailability. It can also handle a much larger number of partitions per cluster.

How it works

KRaft uses multiple quorum controllers to manage Kafka metadata. When the KRaft quorum controllers start up, they designate a leader within the group. The leader is responsible for receiving updates from brokers and making metadata changes. The other quorum controllers, called "followers," replicate the leader's state and metadata changes. This ensures that all quorum controllers have consistent metadata.

When a metadata change occurs, the leader broadcasts the change to all of the follower controllers. The followers acknowledge the change and apply it to their own metadata states. If a follower fails to acknowledge the change, the leader will retry until a quorum of followers acknowledges the change. This process provides a more resilient and fault-tolerant approach to metadata management than the ZooKeeper-based architecture provides.

Other benefits of Kafka with KRaft include:

- **Improved scalability**: KRaft allows Kafka brokers to scale horizontally, allowing for better distribution of workload.
- **Improved message delivery reliability:** KRaft provides more consistent replication and message delivery guarantees, reducing the risk of data loss or corruption.
- **Simplified configuration management**: KRaft reduces the administrative overhead of managing large Kafka deployments.
- **Improved security features:** This new and improved version of Kafka comes with support for TLS 1.3, authentication using OAuth 2.0, and support for Java 11.

How to upgrade to and configure Kafka 3.4 with KRaft

The Apache.org Kafka documentation describes how to upgrade to Kafka version 3.4.0 from previous versions of Kafka through 3.3x, as well as how to upgrade a KRaft-based cluster to 3.4.0 from any version of Kafka from 3.0.x through 3.3.x. Both appear below (copied *mostly* verbatim—with the omission of instructions for upgrading versions prior to 2.1x-for your convenience). If you are upgrading from a version prior to 2.1x, I recommend visiting the documentation link above, since it's a bit more complicated and may impact performance.

For a rolling upgrade to 3.4.0 from a Kafka version using ZooKeeper:

Update `server.properties` on all brokers and add the following properties:

1. Set `inter.broker.protocol.version`=CURRENT_KAFKA_VERSION (CURRENT_KAFKA_VERSION refers to the version you are upgrading from, e.g., 3.3, 3.2, etc.)
2. Upgrade the brokers one at a time: shut down the broker, update the new version of Kafka, then restart it. Once you have done so, the brokers will be running the latest version and you can

verify that the cluster's behavior and performance meets expectations. It is still possible to downgrade at this point if there are any problems.
3. Once the cluster's behavior and performance has been verified, bump the protocol version by editing `inter.broker.protocol.version` and setting it to 3.4.
4. Restart the brokers one by one for the new protocol version to take effect. Once the brokers begin using the latest protocol version, it will no longer be possible to downgrade the cluster to an older version.

Upgrading a KRaft-based cluster to 3.4.0 from any version 3.0.x through 3.3.x

If you are upgrading from a version prior to 3.3.0, please note that once you have changed the `metadata.version` to the latest version, it will not be possible to downgrade to a version prior to 3.3-IV0. Please refer to the Apache documentation for more information.

For a rolling upgrade:

1. Upgrade the brokers one at a time: shut down the broker, update the code, and restart it. Once you have done so, the brokers will be running the latest version and you can verify that the cluster's behavior and performance meets expectations.
2. Once the cluster's behavior and performance has been verified, bump the metadata.version by running `./bin/kafka-features.sh upgrade --metadata 3.4`

Running Kafka with KRaft

Once you've upgraded Kafka to leverage KRaft, it's time to start it up and set up your clusters. Here's the rundown, again, courtesy of the Apache documentation. (Note: Your local environment must have Java 8+ installed.)

Here's how to get started:

First, generate a cluster UUID:

```
$ KAFKA\_CLUSTER\_ID="$(bin/kafka-storage.sh random-uuid)"
```

Next, format the log directories:

```
$ bin/kafka-storage.sh format -t $KAFKA\_CLUSTER\_ID -c config/kraft/server.properties
```

Finally, start the Kafka server:

```
$ bin/kafka-server-start.sh config/kraft/server.properties
```

Once the Kafka server has successfully launched, you will have a basic Kafka environment running and ready to use. From there, you'll set up your topics, write/read events into/from the topics, and continue on your merry way.

We'd love to meet you. Connect with Outshift on Slack!

Top comments (0)

More from Outshift By Cisco

What You Need to Know as a Kafka Developer

#kafka #devops

Install and Maintain Apache Kafka® with a GitOps Approach

#kafka #gitops #devops #argocd

How to Deploy Apache Kafka® on Kubernetes If You're in a Time Crunch

#kafka #kubernetes

DEV Community

Check out this survey and help us moderate our community by becoming a tag moderator here at DEV.