Kafka Raft Mode: запуск Kafka без ZooKeeper

Официально: Kafka Raft (KRaft) скоро заменит ZooKeeper в Apache Kafka. Многие разработчики в восторге от этого изменения, но оно повлияет на команды, которые в настоящее время используют Kafka с ZooKeeper, которым необходимо определить путь обновления, как только ZooKeeper перестанет поддерживаться.

В этом блоге наш эксперт объясняет, что такое Kafka Raft (KRaft), чем реализации Raft отличаются от развертываний на основе ZooKeeper и что следует учитывать при планировании перехода на среду KRaft.

Что такое Kafka Raft (Крафт)?

Почему Kafka Raft заменяет ZooKeeper?

Чем Kafka Raft отличается от ZooKeeper?

График прекращения поддержки ZooKeeper

Переход с ZooKeeper на Kafka Raft

Последние мысли

**Что такое Kafka Raft (Крафт)?**

Kafka Raft (что в широком смысле означает «Надежный, реплицируемый, избыточный и отказоустойчивый») или KRaft — это реализация алгоритма консенсуса Raft, разработанная Kafka.

Протокол Raft Consensus, созданный в качестве альтернативы семейству алгоритмов Paxos, призван стать более простой реализацией консенсуса, цель которой - облегчить понимание, чем Paxos. И Paxos, и Raft работают одинаково при нормальных стабильных условиях эксплуатации, и оба протокола выполняют следующее:

Лидер записывает операцию в свой журнал и просит следующие серверы сделать то же самое.

Операция помечается как «зафиксированная», как только большинство серверов подтверждают операцию.

Основная разница между Raft и Paxos проявляется когда операции идут ненормально и необходимо избрать нового лидера. Оба алгоритма гарантируют, что журнал нового лидера будет содержать самые актуальные коммиты, но то, как они выполняют этот процесс, различается.

На Paxos выборы лидера содержат не только предложение и последующее голосование, но также должны содержать все недостающие записи в журнале, которые отсутствуют у кандидата. Последователи в реализациях Paxos могут голосовать за любого кандидата, и как только кандидат будет избран лидером, новый лидер будет использовать эти данные для обновления своего журнала для поддержания актуальности.

В Raft, с другой стороны, подписчики будут голосовать за кандидата только в том случае, если журнал кандидата не менее актуален, чем журнал подписчика. Это означает, что только самый современный кандидат будет избран лидером. В конечном счете, оба протокола удивительно похожи в своем подходе к решению проблемы консенсуса. Однако, поскольку Raft делает некоторые базовые предположения относительно данных, а именно порядка коммитов в журнале, мы можем увидеть повышение эффективности выборов в Raft.

Что это значит по отношению к Кафке? С протокольной стороны дела не очень. ZooKeeper использует собственный консенсусный протокол под названием ZAB (ZooKeeper Atomic Broadcast), который гораздо больше ориентирован на общий порядок коммитов в журнале изменений. Такое внимание к порядку коммитов позволяет консенсусу Raft вполне вписаться в экосистему Kafka.

Тем не менее, изменения с точки зрения инфраструктуры будут весьма заметными. Поскольку логика Kraft включена в базовый код каждого брокера, узлы ZooKeeper больше не будут частью инфраструктуры Kafka. Обратите внимание, что это не обязательно означает меньшее количество серверов в производственной среде — подробнее об этом позже.

**Почему Kafka Raft заменяет ZooKeeper?**

Чтобы понять, почему руководство сообщества решило отказаться от ZooKeeper, мы можем взглянуть непосредственно на KIP-500 и их аргументы. Короче говоря, этот шаг был призван снизить сложность и более надежно обрабатывать метаданные кластера. Удаление требования к ZooKeeper означает, что больше нет необходимости развертывать две совершенно разные распределенные системы. ZooKeeper имеет другие шаблоны развертывания, инструменты управления и синтаксис конфигурации по сравнению с Kafka. Объединение функциональности в единой системе уменьшит ошибки конфигурации и общую сложность эксплуатации.

Помимо более простых операций, обработка метаданных как отдельного потока событий означает, что одно число, смещение, может использоваться для описания положения члена кластера и быстро обновляться. По сути, это применяет те же принципы, которые используются для производителей и потребителей, к самим членам кластера Kafka.

**Чем Kafka Raft отличается от ZooKeeper?**

В развертывании Kafka на основе ZooKeeper кластер состоит из нескольких узлов-брокеров и кворума узлов ZooKeeper. В этой среде каждое изменение метаданных кластера рассматривается как изолированное событие, не имеющее связи с предыдущими или будущими событиями. Когда изменения состояния передаются в кластер из контроллера кластера, т. е. брокера, отвечающего за отслеживание/выбор лидера раздела, существует вероятность того, что некоторые брокеры не получат все обновления, или устаревшие обновления могут создать условия гонки, как мы видел в некоторых более крупных установках Kafka. В конечном счете, эти точки отказа могут привести к тому, что брокеры окажутся в разных состояниях.

Хотя это и не совсем точно, поскольку все узлы-брокеры могут (и делают) общаться с ZooKeeper.

Напротив, метаданные в KRaft хранятся внутри самого Kafka, а ZooKeeper фактически заменяется кворумом контроллеров Kafka. Узлы контроллера составляют совокупный кворум для выбора активного контроллера, который управляет разделом метаданных. Этот журнал содержит все, что раньше было найдено ZooKeeper: тема, раздел, ISR, данные конфигурации и т. д. — все это будет расположено в этом разделе метаданных.

Используя алгоритм Raft, узлы контроллера будут выбирать лидера без использования внешней системы, такой как ZooKeeper. Лидер, или активный контроллер, будет лидером раздела метаданных и будет обрабатывать все RPC от брокеров.

На диаграмме ниже показано логическое представление реализации новой кластерной среды с использованием KRaft:

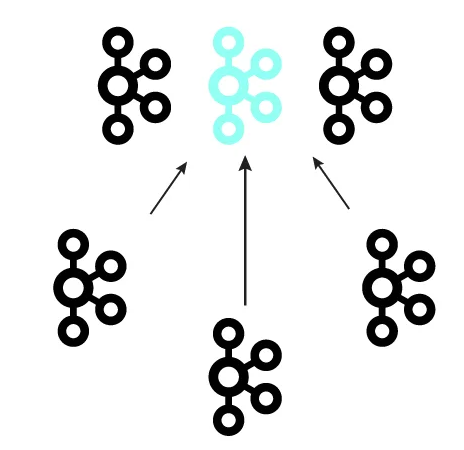


Диаграмма режима Kafka Raft

Обратите внимание, что на диаграмме выше нет двусторонних стрелок. Это указывает на еще одно важное различие между двумя средами: вместо того, чтобы контроллер отправлял обновления брокерам, контроллеры получают метаданные через API MetadataFetch. Подобно обычному запросу на выборку, брокер будет отслеживать смещение последнего полученного им обновления, запрашивая только новые обновления от активного контроллера, сохраняя эти метаданные на диске для более быстрого запуска.

В большинстве случаев брокеру потребуется только запросить дельты журнала метаданных. Однако в случаях, когда данные о брокере отсутствуют или брокер слишком сильно рассинхронизирован, можно отправить полный набор метаданных. Брокер будет периодически запрашивать метаданные, и этот запрос также будет действовать как контрольный сигнал.

Раньше, когда брокер входил в кластер или выходил из него, это отслеживалось в ZooKeeper, но теперь статус брокера будет регистрироваться непосредственно в активном контроллере. В мире после ZooKeeper членство в кластере и обновление метаданных тесно связаны. Отсутствие обновлений метаданных приведет к исключению из кластера.

**График прекращения поддержки ZooKeeper**

Начиная с версии Kafka 3.3, KRaft теперь считается «готовым к производству» (с некоторыми известными оговорками, которые мы обсудим в следующем разделе). Текущий план, согласно KIP-833, заключается в том, что, начиная с версии 3.4, все последующие выпуски 3.X будут считаться «промежуточными выпусками».

Это означает, что, хотя ZooKeeper по-прежнему будет существовать в выпусках 3.X, зависимость от ZooKeeper будет по-прежнему оставаться все более изолированной с упором на переход к реализации KRaft. Это также означает, что для перехода на промежуточный выпуск потребуется обновление выпусков ZooKeeper (Kafka 2.8 и более ранних версий), прежде чем можно будет перейти на выпуск после ZooKeeper (4.X и более поздние версии)

**Переход с ZooKeeper на Kafka Raft**

Тем организациям, которые планируют переход на среду после ZooKeeper, необходимо учитывать немало вещей. В кластере на базе KRaft узлы Kafka могут работать в одном из трех различных режимов, известных какprocess.role. В качестве роли процесса.роли можно установить брокер, контроллер или их комбинацию. В производственном кластере рекомендуется избегать комбинирования процесса.роль — другими словами, выделения ресурсов JVM, назначенных брокерам и контроллерам. Итак, как упоминалось ранее, отказ от ZooKeeper не обязательно означает отказ от вычислительных ресурсов в производстве. Использование комбинированного процесса.роль в средах разработки или промежуточной среды вполне приемлемо.

Кроме того, начиная с версии 3.3 рекомендуется использовать ровно три контроллера независимо от размера кластера в производственных средах. В настоящее время известны проблемы, при которых частичный сбой сети может привести к недоступности метаданных, а наличие более трех контроллеров может увеличить вероятность возникновения проблемы. Сообщество в настоящее время работает над исправлениями этих проблем, которые должны быть доступны в последующих выпусках.

Наконец, в Kafka 3.3 отсутствуют некоторые известные функции:

Настройка пользователей SCRAM через административный API

Поддержка конфигураций JBOD с несколькими каталогами хранения.

Изменение некоторых динамических конфигураций автономного контроллера KRaft.

Токены делегирования

Обновление из режима ZooKeeper

В некоторых случаях эти недостающие функции могут сделать переход на KRaft невозможным. Например, если вы используете токены делегирования при аутентификации брокера/клиента, переход на взаимную аутентификацию TLS/SSL может оказаться значительным шагом вперед. Кроме того, большие и сложные существующие кластеры Kafka могут счесть отсутствие прямого пути обновления из режима ZooKeeper в KRaft довольно устрашающим.

**Последние мысли**

Для новых реализаций использование KRaft определенно не составит труда, но для зрелых сред Kafka переход на версию после ZooKeeper — это полная замена вашего кластера со всеми вытекающими отсюда сложностями. В таких случаях решающее значение имеет создание подробного плана миграции с синими и зелеными стратегиями развертывания. А если вашей команде не хватает опыта работы с Kafka, хорошей идеей будет также обратиться за внешней поддержкой для руководства вашей миграцией.