СБЕР УНИВЕРСИТЕТ

Школа Java Middle Developer

Kafka

Репликация данных

Содержание

1. Сценарии использования
2. Зеркальное копирование

Копирование данных между кластерами Kafka

Сценарии зеркального копирования данных между кластерами

- ✓Региональные и центральные кластеры
- ✓Избыточность
- ✓Миграция в облако

Мультикластерные архитектуры

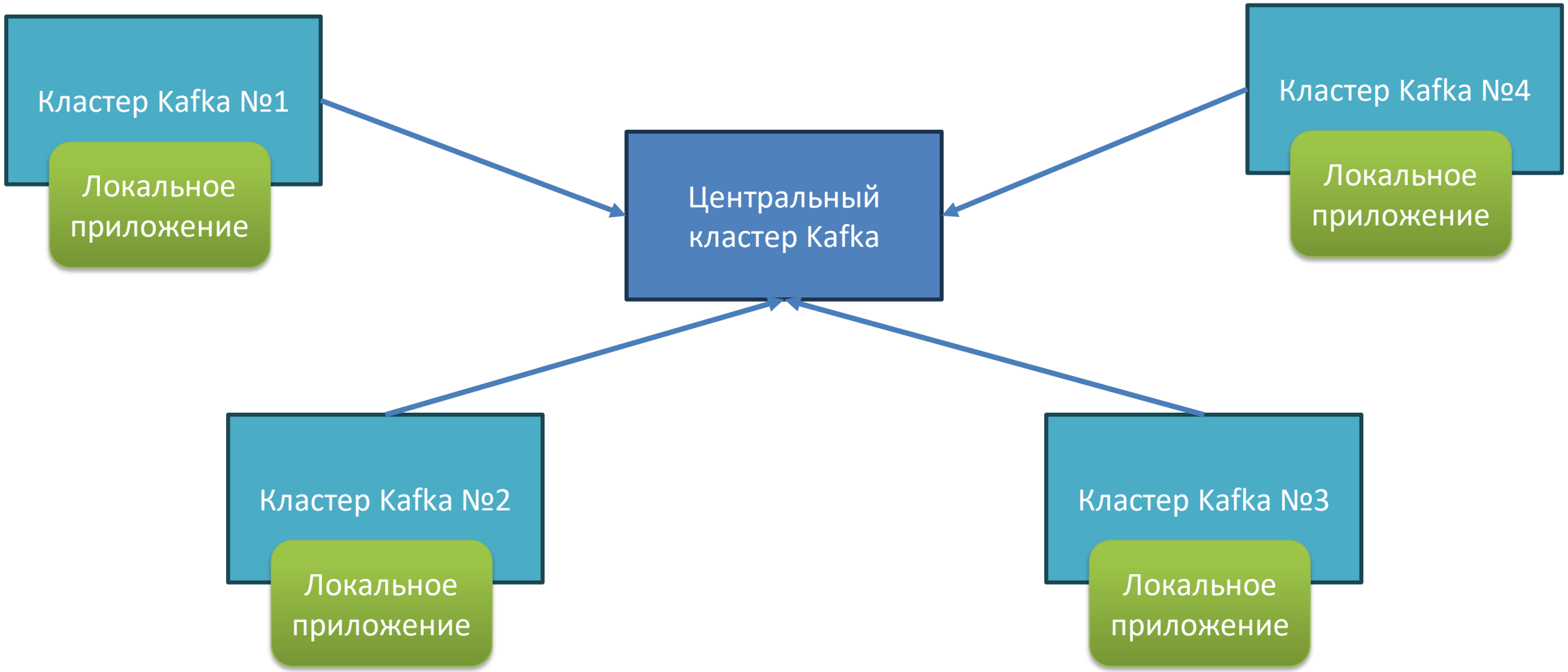
Реалии взаимодействия между различными ЦОД

- Высокая длительность задержек
- Ограниченная пропускная способность сети
- Повышенные по сравнению с работой в пределах одного ЦОД затраты

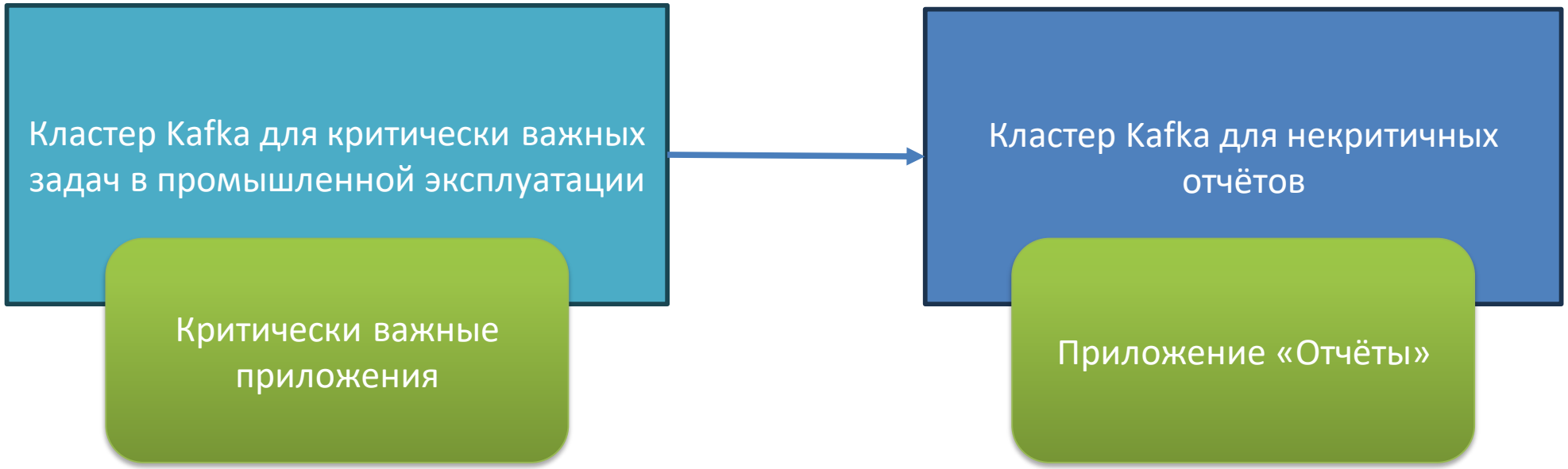
Реалии взаимодействия между различными ЦОД

- Не менее одного кластера на ЦОД
- Каждое событие реплицируется ровно один раз между каждой парой ЦОД
- По возможности предпочитаем потребление данных из удаленного ЦОД отправке данных в него

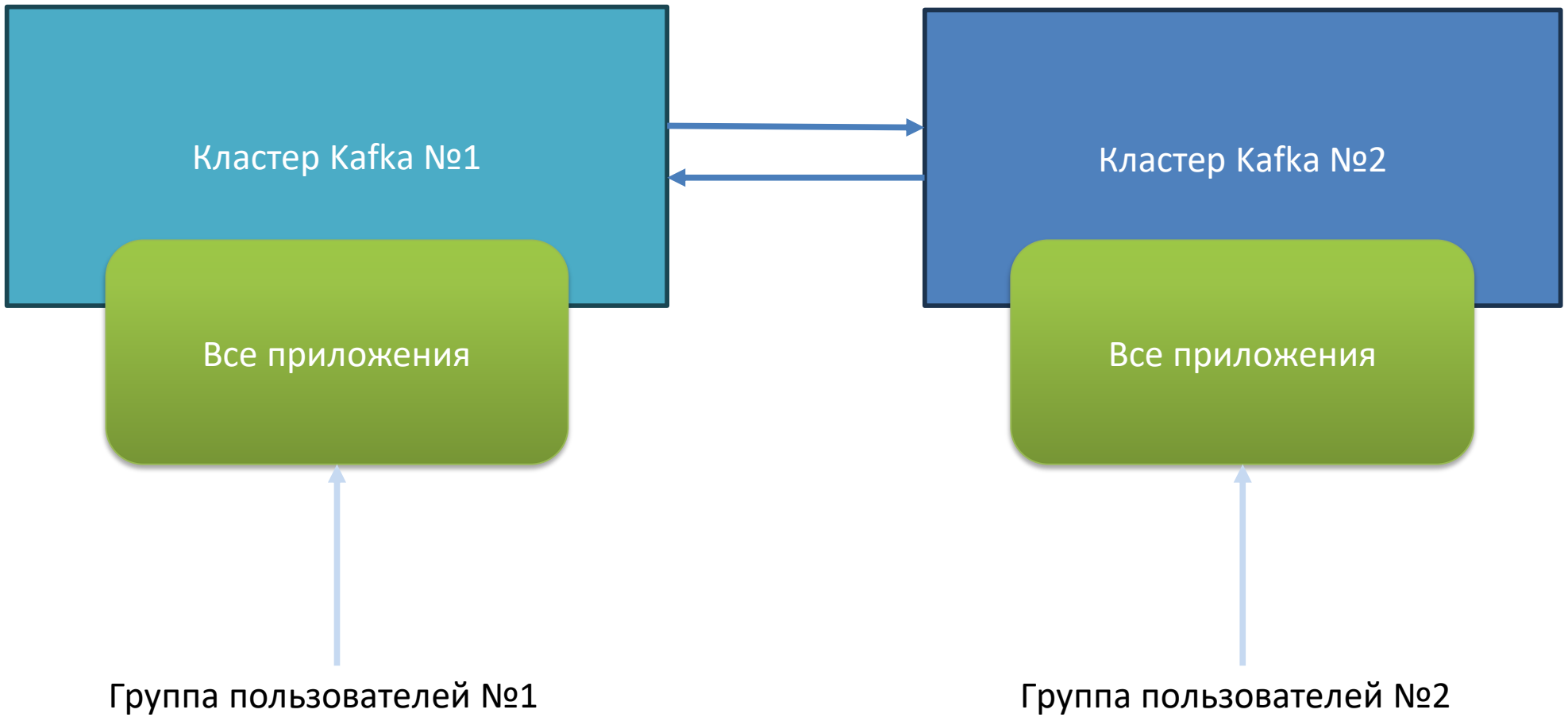
Архитектура с топологией типа «звезда»



Архитектура с топологией типа «звезда»



Архитектура типа «активный - активный»



Архитектура типа «активный - активный»

- При отправке пользователем события в один ЦОД и чтении событий из другого существует вероятность того, что записанное им событие еще не попало во второй ЦОД

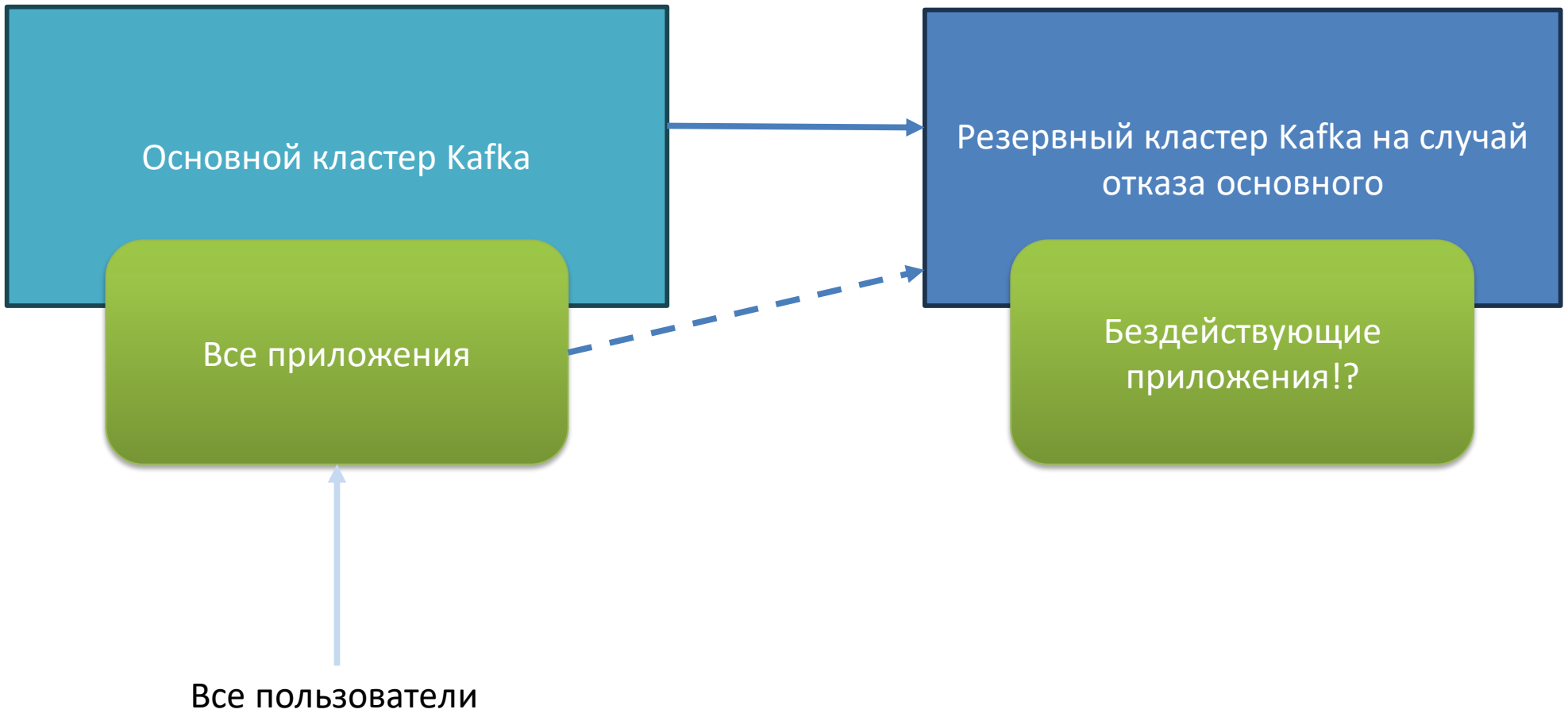
- Событие из одного ЦОД гласит, что пользователь заказал книгу А, а относящееся примерно к тому же моменту времени событие из другого ЦОД - что он же заказал книгу Б

Архитектура типа «активный - активный»

- Часть проблемы зеркального копирования по типу «активный - активный», особенно в случае более чем двух ЦОД, состоит в том, что вам понадобится процесс зеркального копирования для каждой пары ЦОД и каждого направления

- Необходимо избегать зацикливания - бесконечного зеркального копирования одного и того же события туда и обратно

Архитектура типа «активный — резервный»



Архитектура типа «активный — резервный»

- Преимущества: простота настройки и возможность использования практически в любом сценарии

- Недостатки: простаивает отличный кластер, да и переключение с одного кластера Kafka на другой на практике не такая уж простая вещь

Потери данных и несогласованности при внеплановом восстановлении после сбоя

- Все программные решения для зеркального копирования Kafka асинхронны

- Следует всегда контролировать отставание DR-кластера от основного и не позволять ему отставать слишком сильно

Начальное смещение для приложений после аварийного переключения

Варианты решения проблемы чтения данных с нужного места после аварийного переключения:

- Автоматический сброс смещения
- Репликация топика для смещений
- Переключение по времени
- Внешнее хранение соответствий смещений

После аварийного переключения

Очень заманчиво просто изменить направление процесса зеркального копирования и начать зеркально копировать данные из нового основного кластера в старый. Однако при этом возникают два важных вопроса:

- Как узнать, с какого места начинать зеркальное копирование?
- В исходном основном кластере содержатся события, отсутствующие в DR-кластере?

Обнаружение кластеров

- В случае аварийного переключения ваши приложения должны будут откуда-то узнать, как начать взаимодействие с резервным кластером

- Клиентам Kafka достаточно успешно обратиться к одному брокеру, чтобы получить метаданные кластера и обнаружить все остальные

Эластичные кластеры

- Эластичные кластеры (stretch clusters) предназначены для предотвращения отказа кластера Kafka в случае сбоя всего ЦОД
- Эластичные кластеры – не мультикластерные
- Для обеспечения согласованности всех брокеров кластера используется обычный механизм репликации Kafka
- Эластичный кластер можно настроить так, чтобы подтверждение отправлялось после успешной записи сообщения в брокеры Kafka в двух ЦОД (настройки min.insync.replicas и acks=all)

Эластичные кластеры

- Эта архитектура защищает от ограниченного списка типов аварий: только от отказов ЦОД, но не от отказов приложений или Kafka
- Использовать эту архитектуру имеет смысл, если у вас есть возможность установить Kafka по крайней мере в трёх ЦОД с высокой пропускной способностью и низкой сетевой задержкой взаимодействия между ними

Утилита MirrorMaker (Apache Kafka)

- MirrorMaker - простая утилита Apache Kafka для зеркального копирования данных между двумя ЦОД
- Представляет собой набор потребителей, состоящих в одной группе и читающих данные из выбранного набора топиков для репликации
- У каждого из процессов MirrorMaker имеется один производитель

Настройка MirrorMaker

bin/kafka-mirror-maker.sh --consumer.config config/consumer.properties

--producer.config config/producer.properties --new.consumer --num.streams=2

--whitelist "."*

- consumer.config - настройки для всех потребителей, извлекающих данные из исходного кластера
- producer.config - настройки производителей, которые используются MirrorMaker для записи в целевой кластер
- num.streams – количество потоков, каждый из которых представляет один потребитель, читающий данные из исходного кластера
- whitelist - регулярное выражение для названий зеркально копируемых топиков

uReplicator компании Uber

uReplicator появился в результате решения проблем:

- Задержки из-за перебалансировки: добавление потоков и экземпляров MirrorMaker, перезапуск экземпляров MirrorMaker или добавление новых топиков, соответствующих регулярному выражению из параметра whitelist, - всё это приводит к перераспределению потребителей
- Сложности при добавлении топиков: использование регулярного выражения в качестве whitelist топиков означает, что MirrorMaker придется выполнять перераспределение при каждом добавлении соответствующего топика в исходный кластер

Replicator компании Confluent

Replicator появился в результате решения проблем:

- Расхождения конфигураций кластеров. MirrorMaker обеспечивает согласованность только данных в исходном и целевом кластерах. В результате у топиков могут появиться различное число партиций, разные коэффициенты репликации и различные настройки уровня топика
- Проблемы управления кластером. MirrorMaker обычно развертывается в виде кластера из нескольких экземпляров. То есть это ещё один кластер, который нужно как-то развернуть, управлять им и выполнять мониторинг

Спасибо за внимание