Эксплуатация Arenadata Streaming (Kafka, NiFi)

ARENADATA

Архитектура и инструменты Apache Kafka: Brocker, Connect, REST Proxy, Schema Regestry, ksqlDB

Agenda

- Базовые концепции и объекты Kafka:
  Consumers, Producers, Topics, Partitions, Segments, Replication, Offset, Messages, consumer groups, Brokers, Schemas
- Topics Kafka:
  основные операции (создание, управление, мониторинг, console CLI, partitions, replication, compaction, retention). Гарантии надежности Kafka (доставка/потребление).
  Лабораторная работа.
- Zookeeper. Выбор лидера
- Producers Kafka:
  Запись сообщений в Kafka, console Producer. Настройка Producers.
- Consumers Kafka:
  Consumer groups, ребалансировка разделов. Настройка Consumers. Изменение параметров Topics, consumer groups, Partitions.
- Kafka Connect:
  Основные понятия и инструменты. FileStream Connectors. CDC Debezium. Kafka ADB Connectors.
- Kafka REST Proxy, ksqlDB:
  Основные возможности и примеры использования.
- Мультикластерные архитектуры:
  Топология мультикластерной репликации. Настройка MirrorMaker, Mirror Connectors.

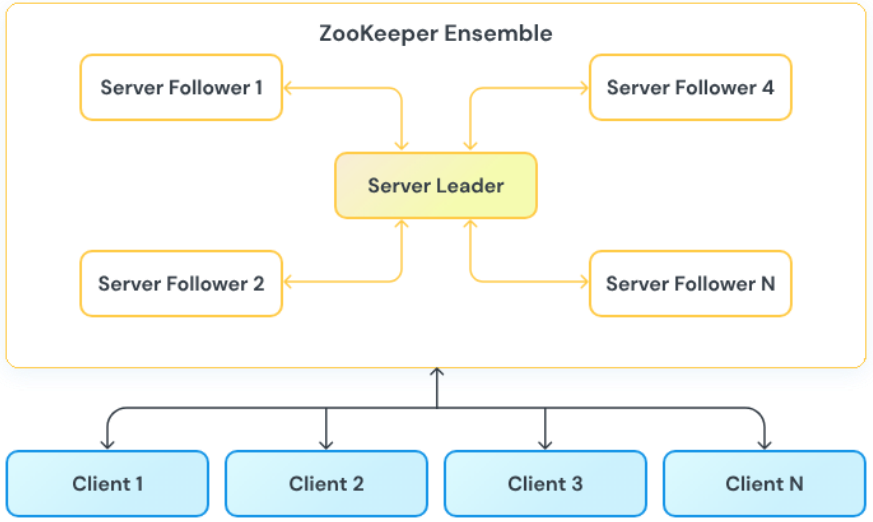
Zookeeper. Выбор лидера

ARENADATA

Архитектура

Zookeeper использует архитектуру клиент-сервер и содержит следующие компоненты:

- **Ensemble** — это группа (ансамбль) серверов ZooKeeper в количестве не менее трех для обеспечения надежности. Когда один сервер становится недоступным, оставшиеся два образуют кворум (quorum), который продолжает работу кластера. Рекомендуется использовать нечетное количество серверов в кластере, то есть: 3, 5, 7 и так далее.
- **Server** — один из участников ансамбля Zookeeper. Его назначение состоит в предоставлении всех видов сервиса, необходимых клиентам. Существуют два типа серверов:
- **Server Leader** — лидер, обеспечивает автоматическое восстановление работоспособности кластера в случае сбоев на серверах другого типа. Серверы выбирают лидера при начальном запуске ZooKeeper.
- **Server Follower** — ведомый сервер, которым является каждый сервер ансамбля, кроме лидера. Он следует указаниям лидера.
- **Client** — клиент, являющийся потребителем сервисов ZooKeeper.



Типы znodes

Persistent	Тип znodes, используемый по умолчанию. Znodes такого типа продолжают существовать, даже если создавший их клиент не подключен к серверам кластера
Ephemeral	Тип znodes, время существования которых определяется продолжительностью клиентской сессии (в рамках которых они созданы). Ephemeral znodes не могут иметь дочерних znodes
Sequential	Когда клиент запрашивает создание znode такого типа, ZooKeeper добавляет к указанному полному пути znode 10-значный последовательный номер, например /top/myspace/mynode0000000001. Sequential znodes играют важную роль в блокировании и синхронизации процессов. Sequential znodes являются одновременно persistent или ephemeral. Это означает, что существует не один тип, а два: persistent-sequential и ephemeral-sequential
Container	Тип znodes, созданный для упрощения выполняемой ZooKeeper очистки от неиспользуемых более znodes (znodes, оставшихся без дочерних нод). Container-нода аналогична Persistent-ноде, но с дополнительной функцией постановки в кандидаты на удаление ZooKeeper-сервером, как только удаляется ее последняя дочерняя znode

Выбор контроллера Kafka

Контроллер Kafka — брокер, который отвечает за:

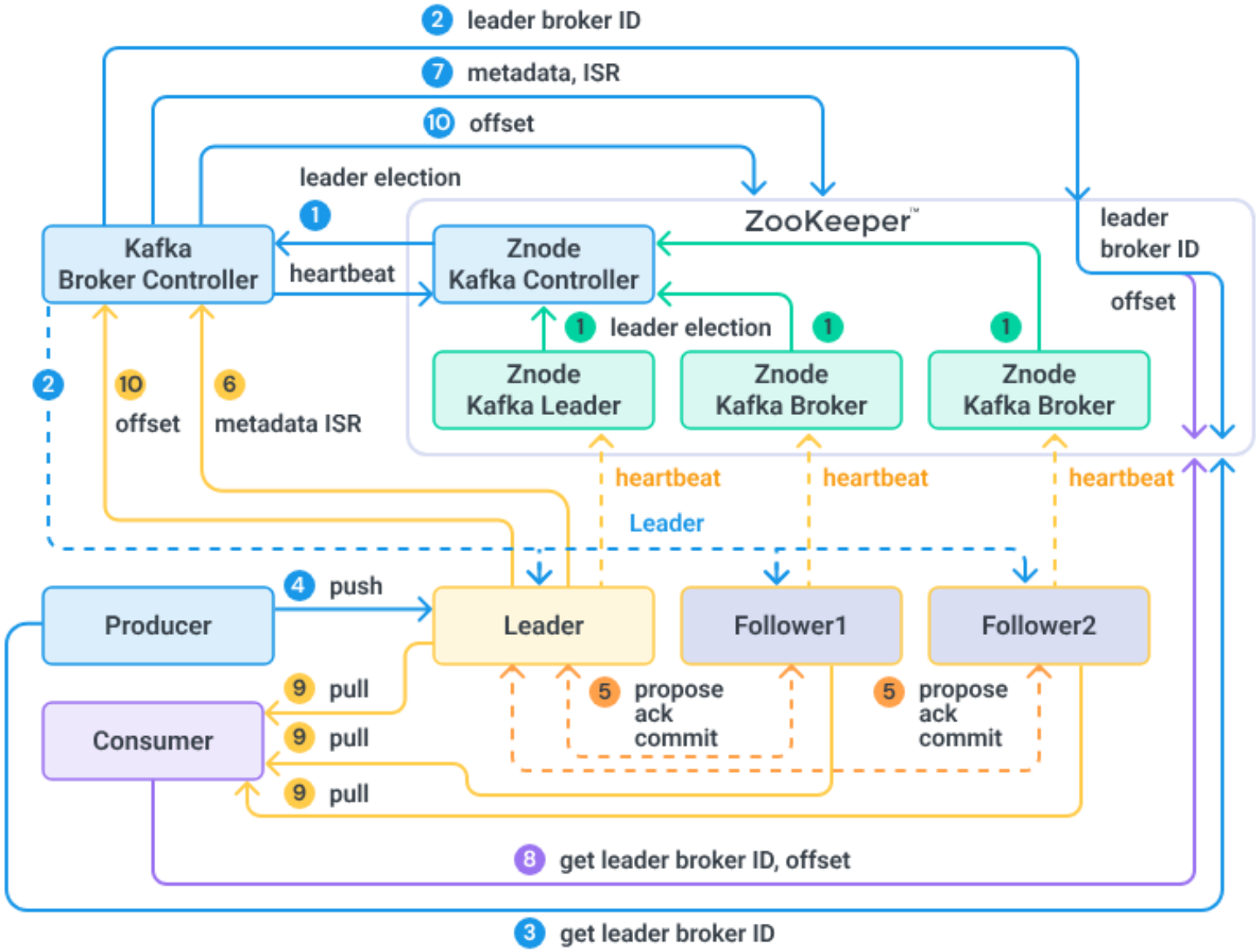
- ведение списка синхронизированных реплик (ISR);
- выбор нового лидера журнала (партиции) из ISR, когда текущий лидер выходит из строя;
- управление партициями;
- назначение партиций пользователям.

Контроллером может являться только один брокер одновременно. Выбор контроллера выполняется автоматически при помощи ZooKeeper в следующей последовательности:

- Каждый брокер пытается создать эфемерный znode с наименованием /controller в ZooKeeper.
- Первый брокер, создавший этот эфемерный znode, возьмет на себя роль контроллера, и каждый последующий запрос брокера будет получать сообщение node already exists.
- После того, как контроллер установлен, ему назначается "эпоха контроллера".
- Текущая эпоха контроллера передается по всему кластеру, и если брокер получает запрос контроллера от более старой эпохи контроллера, он игнорируется.
- Если происходит сбой контроллера, происходит новый выбор и создается новая эпоха контроллера, которая передается в кластер.

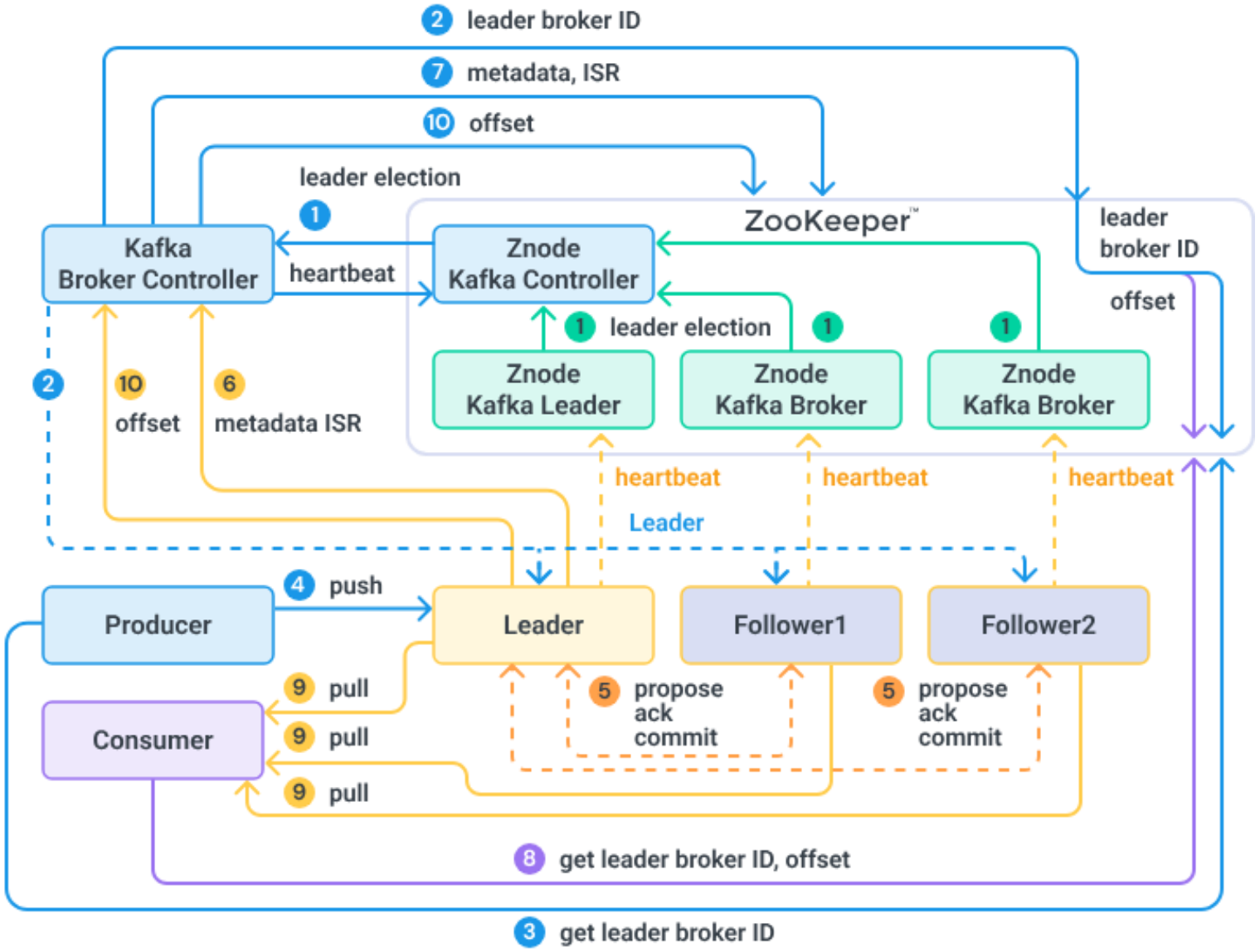
Лидер реплицированного журнала Kafka

Лидер (Leader) журнала (партиции) — брокер, который работает с клиентами. К лидеру осуществляют запросы последователи (followers) — брокеры, которые хранят реплику всех данных партиций. Потребители могут читать сообщения как с лидера, так и с последователей.



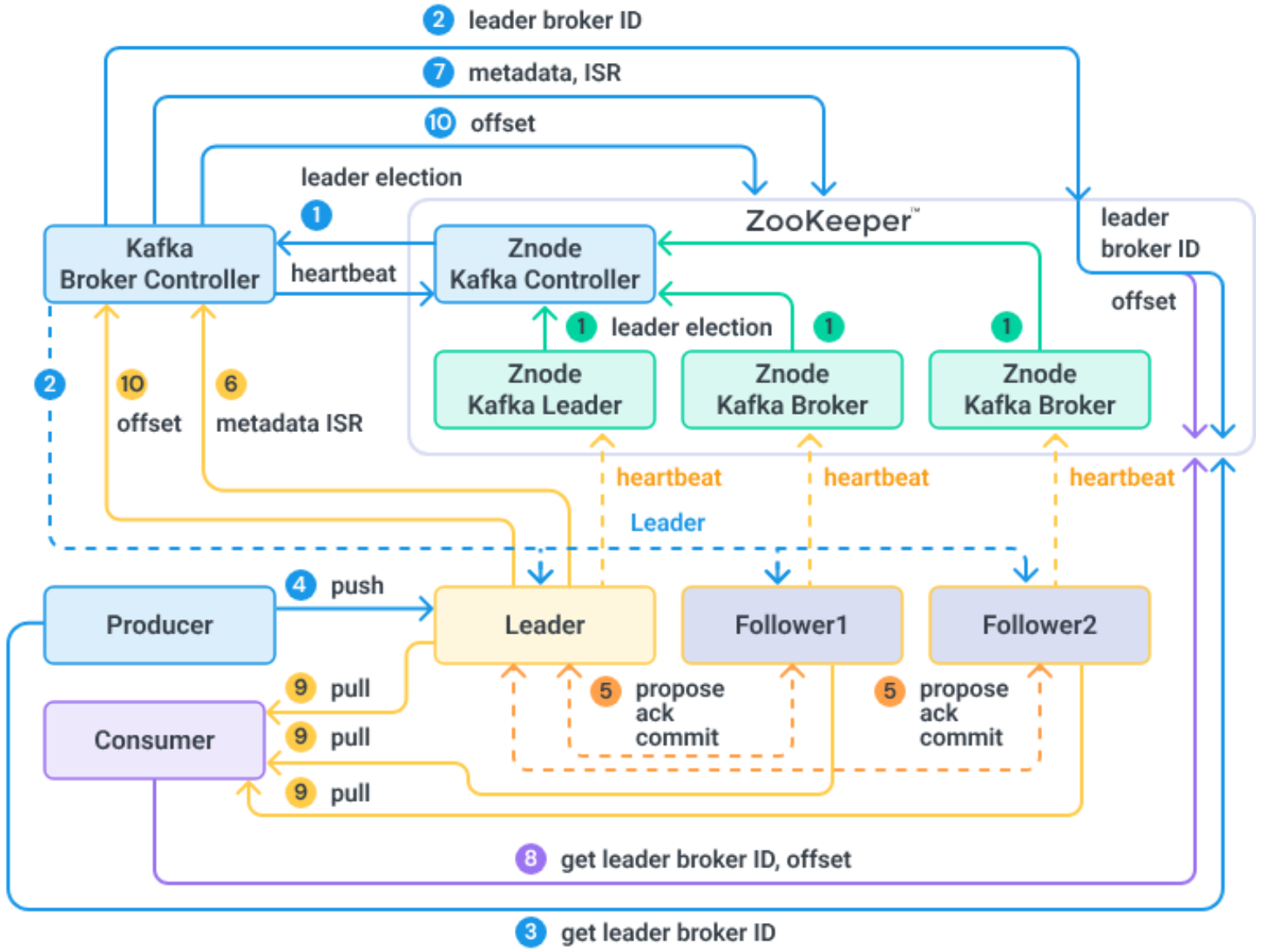
Лидер реплицированного журнала Kafka

1. Активными считаются брокеры, эфемерные znode которых созданы в ZooKeeper и которые посылают контрольные сигналы (heartbeat) на сервер ZooKeeper для поддержания сессии. Для отслеживания состояния брокеров нода контроллера подписывается (watches) на ноды брокеров.
2. ID нового лидера, контроллер записывает информацию о новом лидере в ZooKeeper и отправляет данные каждому брокеру, который размещает реплику этой партиции.
3. Производитель (producer) подключается к ZooKeeper и получает ID лидера партиции, куда он будет записывать данные.
4. Производитель записывает сообщение в топик на лидера партиции.
5. Взаимодействие между лидером партиции и последователями разделяется на три фазы:
   - propose — лидер предлагает создать репликацию последователям.
   - ack — лидер получает ответ от последователей о создании реплики.
   - commit — лидер фиксирует создание ISR при получении сообщения о создании реплики от всех последователей. ISR-реплики — являются кандидатами в лидеры при отказе действующего лидера.



Лидер реплицированного журнала Kafka

6. Лидер передает данные о записанной партиции (метаданные) и о созданных ISR в контроллер.
7. Контроллер записывает информацию о созданных ISR в хранилище ZooKeeper, а также метаданные партиции.
8. Потребитель (consumer) подключается к ZooKeeper и получает ID лидера или последователей для запрашиваемой партиции, а также данные о смещении (offset), с которого следует читать.
9. Потребитель запрашивает (pull) сообщения у лидера или последователей.
10. Лидер получает данные о смещении (offset) и передает их в ZooKeeper.



Producers Kafka

Запись сообщений в Kafka, console Producer.
Настройка Producers

ARENADATA

Настройки Producers

- Установить auto.create.topics.enable в значение false (в ADS 3.X+ по умолчанию). Гарантирует упорядоченное создание тем, а не при любом запуске производителя, отправляющего сообщение в новую тему.
- retry.backoff.ms определяет задержку в миллисекундах между повторными попытками отправки сообщений производителем.
- batch.size определяет максимальный размер пакета данных, который производитель будет собирать перед отправкой брокеру. Это значение измеряется в байтах и влияет на количество данных, которые производитель будет пытаться отправить за один раз. Меньший размер пакета может привести к более частым отправкам, но с меньшим объемом данных, в то время как больший размер пакета позволяет отправлять больше данных за один раз, но может увеличить задержку.
- buffer.memory в Apache Kafka определяет количество памяти, которое будет выделено производителем для буферизации ожидающих отправки сообщений. Если производитель генерирует сообщения быстрее, чем они могут быть отправлены брокеру, эти сообщения будут временно храниться в этом буфере.
- Основные параметры конфигурации производителя:
- key.serializer и value.serializer: Классы сериализации, которые определяют, как ключи и значения сообщений будут преобразованы в байты перед отправкой.
- linger.ms: Время ожидания в миллисекундах перед отправкой пакета данных, даже если он не достиг batch.size.
- acks: Уровень подтверждения от брокеров, который требуется производителю перед считать сообщение отправленным.
- max.block.ms: Максимальное время ожидания в миллисекундах, в течение которого производитель будет блокироваться, если буфер заполнен.
- retries и retry.backoff.ms: Количество попыток и задержка между попытками повторной отправки сообщения после сбоя.

Consumers Kafka

Consumer groups, ребалансировка разделов. Настройка Consumers. Изменение параметров Topics, consumer groups, Partitions

ARENADATA

Настройки Consumers

- group.id – Идентификатор группы Consumer.
- offset – смещение (earliest – начало, latest – d, «число» - номер смещения)
- key.deserializer – Класс для десериализации ключей сообщений.
- value.deserializer – Класс для десериализации значений сообщений.
- auto.offset.reset – Стратегия чтения сообщений (earliest, latest, none).
- enable.auto.commit – Автоматическое сохранение смещений в Kafka.
- auto.commit.interval.ms – Интервал автоматического коммита смещений.
- max.poll.records – Максимальное количество записей для одного вызова poll().
- fetch.min.bytes – Минимальное количество байт для запроса данных.
- fetch.max.wait.ms – Максимальное время ожидания данных от сервера.
- max.partition.fetch.bytes – Максимальный размер данных для запроса к серверу.
- retry.backoff.ms – Время ожидания перед повторной обработкой сообщения при ошибке.
- partition.assignment.strategy – Выбор стратегии ребалансировки задается

Ребалансировка разделов

Ребалансировка партиций (partition.assignment.strategy) — перераспределение разделов топиков между Consumer’s в группе. Необходимо для обеспечения равномерного распределения нагрузки и для того, чтобы в случае выхода из строя одного из Consumer’s, его разделы могли быть обработаны другими участниками группы.

Когда возникает ребалансировка:

- При добавлении нового Consumer в группу.
- При выходе Consumer из группы (сбой/close).
- При добавлении новых разделов в топик, к которому подписаны Consumer’s.
- При изменении количества разделов топика.

Ребалансировка:

- **Остановка обработки сообщений:** Consumer’s останавливают чтение сообщений из разделов.
- **Групповой координатор:** Один из брокеров Kafka выступает в роли группового координатора и управляет процессом ребалансировки.
- **Join Group:** Consumer'ы отправляют запрос на присоединение к группе (Join Group) координатору.
- **Синхронизация:** Координатор выбирает одного из Consumer’s в качестве лидера для выполнения ребалансировки и отправляет ему полный список Consumer’s и разделов. Лидер создаёт новое распределение разделов и отправляет его обратно координатору.
- **Применение нового распределения:** Координатор отправляет новое распределение всем Consumer'ам в группе.
- **Возобновление обработки:** Consumer’s начинают чтение сообщений из новых разделов согласно новому распределению.
- Этот процесс может привести к кратковременной задержке в обработке сообщений, так как во время ребалансировки Consumer’s не читают сообщения. Однако, благодаря ребалансировке, Kafka обеспечивает высокую доступность и масштабируемость потребления сообщений.

Ребалансировка разделов

Стратегии ребалансировки:

- **Range Assignor (Диапазонный Распределитель):** Разделы распределяются между Consumer’s равномерно на основе диапазона. Если у Consumer’s есть сортированный список разделов, каждый Consumer получает последовательный диапазон разделов.
- **RoundRobin Assignor (Распределитель “По Кругу”):** Разделы распределяются между Consumer’s поочерёдно, что обеспечивает равномерное распределение разделов, даже если количество разделов не делится нацело на количество Consumer’s.
- **Sticky Assignor:** Этот метод стремится минимизировать изменения в распределении разделов при ребалансировке. Consumer'ы сохраняют свои разделы, если это возможно, даже при изменениях в группе.
- **CooperativeSticky Assignor:** Улучшенная версия Sticky Assignor, которая выполняет инкрементальную ребалансировку (Consumer’s могут продолжать обрабатывать некоторые разделы во время ребалансировки, что сокращает простои), минимизируя задержки и изменения в распределении разделов.

Schema’s compatibility.level

- http://<schema-registry-host>:<port>/config

{"compatibilityLevel":"BACKWARD"}

Политики чтения схемы:

- **Прямая совместимость (Forward compatibility)**: Потребители могут читать данные, записанные с использованием новой схемы, даже если они были сконфигурированы для старой схемы. Это гарантирует, что новые версии схемы не нарушат работу существующих потребителей.
- **Обратная совместимость (Backward compatibility)**: Потребители, использующие новую версию схемы, могут читать данные, записанные со старыми версиями схемы. Это позволяет обновлять схемы без риска потери совместимости с уже записанными данными.
- **Полная совместимость (Full compatibility)**: Обеспечивает и прямую, и обратную совместимость. Это самый строгий вариант, который гарантирует, что данные всегда будут читаемы независимо от того, какая версия схемы использовалась при записи или чтении.
- **Нет совместимости (None compatibility)**: Не требует никакой совместимости между версиями схемы. Это может быть полезно в случаях, когда совместимость не является приоритетом, например, во время разработки.

Schema’s compatibility.level

- http://<schema-registry-host>:<port>/config

{"compatibilityLevel":"BACKWARD"}

```
{
  "type": "record",
  "name": "Order",
  "fields": [
    {"name": "id", "type":
"string"},
    {"name": "amount", "type":
"double"}
  ]
}
```

```
{
  "type": "record",
  "name": "Order",
  "fields": [
    {"name": "id", "type": "string"},
    {"name": "amount", "type": "double"},
    {"name": "custName", "type": ["null", "string"], "default": null}
  ]
}
```

- Производитель начинает отправлять сообщения с новым полем custName.
- Старые потребители продолжают работать, игнорируя новое поле, так как они используют старую схему.
- Новые потребители могут читать как старые, так и новые сообщения, используя обновленную схему.

Consumer/Producer Schemas Registry

Список всех схем:

http://<shcema_registry_host>:8081/schemas


```
…
{
    "subject": "ads-a-XX-et.testDB.dbo.customers-key",
    "version": 1,
    "id": 4,
    "schema":
"{\"type\":\"record\",\"name\":\"Key\",\"namespace\":\"ads_a_10_et.testDB.dbo.customers\",\"fields\":[{\"name\":\"id\",\"type\":\"int\"}],\"connect.name\":\"ads_a_XX_et.testDB.dbo.customers.Key\"}"
  },
 …
```

Consumer/Producer Schemas Registry

Указать конкретную версию схемы:

```
kafka-console-consumer.sh --bootstrap-server localhost:9092 --topic your-topic \
--property value.deserializer=io.confluent.kafka.serializers.KafkaAvroDeserializer \
--property schema.registry.url=http://localhost:8081 \
--property value.schema.version=1111 \
--from-beginning


kafka-avro-console-producer.sh --broker-list localhost:9092 --topic your-topic \
--property value.serializer=io.confluent.kafka.serializers.KafkaAvroSerializer \
--property schema.registry.url=http://localhost:8081 \
--property value.schema.version=1111
```

Указать схему:

```
kafka-avro-console-producer \
--broker-list localhost:9092 --topic test \
--property
value.schema='{"type":"record","name":"User","fields":[{"name":"name","type":"string"},{"name":"favorite_number","type":["int","null"]},{"name":"
favorite_color","type":["string","null"]}]}'
```

Kafka API’s

- Producer API – позволяет приложению публиковать поток сообщений в один или несколько топиков платформы Kafka.
- Consumer API – позволяет приложению подписываться на один или несколько топиков и обрабатывать принадлежащие им потоки записей.
- Streams API – позволяет приложению выступать в качестве stream processor (потокового процессора). Streams API потребляет входной поток данных из одного или нескольких топиков и создает выходной поток данных так же для одного или нескольких топиков. Таким образом Streams API эффективно преобразует входные потоки в выходные.
- Admin Client API – это Java-based интерфейс, который позволяет управлять и мониторить кластером Kafka. Admin Client API позволяет выполнять административные задачи:
  - Создание/изменение/удаление топиков.
  - Получение информации о кластере (топики, партиции и их состояние, …).
  - Управление группами потребителей.
- Connect API – позволяет реализовывать коннекторы, которые в непрерывном режиме извлекают данные из какой-либо системы исходных данных в Kafka или передают данные из Kafka в какую-либо систему-приемник данных.

Kafka Connect

Основные понятия и инструменты. FileStream Connectors

ARENADATA

Kafka Connect

- Kafka Connect — это инструмент для непрерывной передачи данных между Kafka и другими системами (СУБД, хранилища данных, индексы поиска, файловые системы и т.д.). Предназначен для упрощения процесса добавления новых источников данных в Kafka и потребления данных из Kafka в другие системы.
- Архитектура Kafka Connect:
- **Connectors**:
  - Это плагины, которые предоставляют интерфейс для взаимодействия с источниками данных (Source Connectors) и приемниками данных (Sink Connectors): Source Connectors извлекают данные из источников и передают их в Kafka. Sink Connectors принимают данные из Kafka и передают их в целевые системы.
  - Управляют **Tasks**, которые фактически выполняют работу по перемещению данных. Каждый коннектор может иметь одну или несколько задач.
- **Workers**: Это процессы, которые исполняют задачи, связанные с коннекторами. Worker может быть одиночным процессом (standalone mode) для разработки и тестирования или распределенным (distributed mode) для масштабируемой и отказоустойчивой работы в PROD. В распределенном режиме workers координируются через Kafka, что обеспечивает автоматическое балансирование нагрузки и восстановление после сбоев.
- **Tasks** — это рабочие процессы, которые выполняют передачу данных. Задача может быть источником (Source Task), который читает данные из внешней системы и пишет их в Kafka, или потребителем (Sink Task), который читает данные из Kafka и пишет их во внешнюю систему.
- Kafka Connect использует REST API для управления коннекторами и задачами, что позволяет легко интегрировать его с различными системами управления и мониторинга.

FileStream Connectors

- Starting with version 6.2.1, the FileStream Sink and Source connector artifacts have been moved out of Kafka Connect.
- Confluent Platform and Apache Kafka compatibility:

Confluent Platform	Apache Kafka®	Release Date	Standard End of Support	Platinum End of Support
7.6.x	3.6.x	February 9, 2024	February 9, 2026	February 9, 2027
7.5.x	3.5.x	August 25, 2023	August 25, 2025	August 25, 2026
7.4.x	3.4.x	May 3, 2023	May 3, 2025	May 3, 2026
7.3.x	3.3.x	November 4, 2022	November 4, 2024	November 4, 2025
7.2.x	3.2.x	July 6, 2022	July 6, 2024	July 6, 2025
7.1.x	3.1.x	April 5, 2022	April 5, 2024	April 5, 2025
7.0.x	3.0.x	October 27, 2021	October 27, 2023	October 27, 2024
6.2.x	2.8.x	June 8, 2021	June 8, 2023	June 8, 2024
6.1.x	2.7.x	February 9, 2021	February 9, 2023	February 9, 2024

- Поддерживаемые сервисы ADS

Kafka	Kafka Broker	3.3.2
Kafka Connect	Kafka Connect Worker	3.3.2

- Замена: FileStream Sink and Source connector -> Spool Dir Connectors

FileStream Connectors

- Добавление FileStream Connectors:

1. sudo mkdir /etc/kafka/plugins/
2. sudo cp /usr/lib/kafka-connect/libs/connect-file-*.jar /etc/kafka/plugins/
3. ADCM -> Restart Kafka Connect Service
4. http://ads-a-XX-node-1:8083/connector-plugins:

```
…
  {
    "class": "org.apache.kafka.connect.file.FileStreamSinkConnector",
    "type": "sink",
    "version": "3.3.2"
  },
  {
    "class": "org.apache.kafka.connect.file.FileStreamSourceConnector",
    "type": "source",
    "version": "3.3.2"
  },
..
```

- Добавление настроек и запуск коннектора:

1. mkdir /tmp/connect_source
2. Создать топик: connect-distributed-test-topic
3. Создать группу: connect-local-file-source-distributed
4. curl -X POST -H "Content-Type: application/json" --data '{

```
  "name": "local-file-source-distributed",
  "config": {
    "connector.class": "org.apache.kafka.connect.file.FileStreamSourceConnector",
    "tasks.max": "1",
    "file": "/tmp/connect_source/file.txt",
    "topic": "connect-distributed-test-topic",
    "group.id": "connect-local-file-source-distributed",
    "key.converter": "org.apache.kafka.connect.storage.StringConverter",
    "value.converter": "org.apache.kafka.connect.storage.StringConverter"
  }
}' http://localhost:8083/connectors
```

5. curl -X DELETE http://localhost:8083/connectors/local-file-source-distributed

Kafka REST Proxy, ksqlDB

Основные возможности и примеры использования

ARENADATA

Kafka REST Proxy

- Kafka REST Proxy — инструмент работа с топиками Kafka по интерфейсу RESTful

Kafka REST Proxy используется для:

- интеграции с веб-приложениями и другими системами, которые не могут напрямую использовать клиенты Kafka;
- тестирования, отладки приложений.

Примеры RESTful запросов:

- Чтение сообщений из топика: POST /consumers/(string:group_name)

```
curl -X POST -H "Content-Type: application/vnd.kafka.v2+json" \
      --data '{"name": "my_consumer", "format": "json", "auto.offset.reset": "earliest"}' \
      http://hostname:8082/consumers/my_group
```

- Список топиков: GET /topics
- Конфигурационные параметры топиков: GET /topics/(string:topic_name).
- Партиции: GET /topics/(string:topic_name)/partitions.

Kafka KSQL

- Kafka KSQL — KSQL является потоковым SQL-диалектом, разработанным для Kafka, который позволяет выполнять аналитические запросы над потоками данных в Kafka

Примеры ksql:

```
kafka-topics.sh --create --topic orders_topic --bootstrap-server localhost:9092 --partitions 3 --replication-factor 1

sudo ksql http://ads-a-XX-node-1.ru-central1.internal:8088

>CREATE STREAM orders_stream (order_id INT, item_id INT, quantity INT, price DECIMAL(5, 2))

>WITH (KAFKA_TOPIC='orders_topic', VALUE_FORMAT='JSON', PARTITIONS=3);

Message

----------------

 Stream created

----------------

ksql> INSERT INTO orders_stream (order_id, item_id, quantity, price) VALUES (1, 101, 2, 49.99);

ksql> quit

kafka-console-consumer.sh --topic orders_topic --bootstrap-server localhost:9092 --from-beginning

{"ORDER_ID":1,"ITEM_ID":101,"QUANTITY":2,"PRICE":49.99}
```

Мультикластерные архитектуры

Топология мультикластерной репликации. Настройка MirrorMaker, Mirror Connectors

ARENADATA

Архитектуры

Мультикластерные арихитектуры (Confluent):

- **Active/Passive**:
  - Один кластер активно обрабатывает данные, в то время как другой находится в режиме ожидания.
  - В случае сбоя активного кластера, пассивный кластер может быстро переключиться на обработку данных.
- **Active/Active**:
  - Оба кластера активно обрабатывают данные и синхронизируются друг с другом.
  - Это обеспечивает высокую доступность и балансировку нагрузки.
- **Stretch Clusters (Растянутые кластеры)**:
  - Кластеры распределены по разным географическим регионам.
  - Это обеспечивает географическую отказоустойчивость и близость к потребителям данных.
- **Multi-Region Clusters** :
  - Кластеры расположены в разных регионах и работают как единая система.
  - Это обеспечивает высокую доступность и устойчивость к региональным сбоям.
- ...

MM2

- MirrorMaker 2.0 — основанный на платформе сервиса Kafka Connect механизм репликации данных из исходного кластера на удаленный.

Компоненты и принципы работы MM2:

- **Коннекторы**:
  - MM2 использует коннекторы Kafka Connect для репликации топиков, потребителей и продюсеров между кластерами.
  - Коннекторы можно настроить для однонаправленной или двунаправленной репликации.
- **Топик репликации**:
  - Для каждого топика в исходном кластере создается соответствующий топик репликации в целевом кластере.
  - Имена топиков репликации обычно содержат префикс, указывающий на исходный кластер.
- **Синхронизация потребителей**:
  - MM2 синхронизирует смещения потребителей между кластерами, позволяя потребителям переключаться между кластерами без потери данных.
- **Автоматическое обнаружение топиков**:
  - MM2 может автоматически обнаруживать и реплицировать новые топики без необходимости ручной настройки.
- **Управление схемами**:
  - Если используется Schema Registry, MM2 может синхронизировать схемы между кластерами.

Mirrors Connectors

Connectors:

- MirrorSourceConnector осуществляет репликацию топиков из исходного кластера в целевой кластер.
- MirrorCheckpointConnector создает контрольные точки смещения потребителя и синхронизирует смещение со служебным топиком __consumer_offsets исходного кластера.
- MirrorHeartbeatConnector периодически проверяет подключение между кластерами, создавая сообщения в специальном топике heartbeats в исходном кластере через заданный период времени и считывая их в целевом кластере.

http://<kafka_connect_worker>:8083/connector-plugins


```
[
  {
    "class": "org.apache.kafka.connect.mirror.MirrorCheckpointConnector",
    "type": "source",
    "version": "3.3.2"
  },
  {
    "class": "org.apache.kafka.connect.mirror.MirrorHeartbeatConnector",
    "type": "source",
    "version": "3.3.2"
  },
  {
    "class": "org.apache.kafka.connect.mirror.MirrorSourceConnector",
    "type": "source",
    "version": "3.3.2"
  }
]
```