



OTUS

# Онлайн образование

[otus.ru](http://otus.ru)

• REC

Проверить, идет ли запись

Меня хорошо видно  
&& слышно?



Тема вебинара

# Установка и настройка кластера



**Заигрин Вадим**

Ведущий эксперт по технологиям, Сбербанк

vzaigrin@yandex.ru

<https://t.me/vzaigrin>

# Преподаватель



## Вадим Заигрин

Более 30 лет в ИТ:

- Big Data
  - Data Engineer
  - Data Science
- Разработка
  - Scala, Java, Python, C, Lisp
- IT Infrastructure
  - Администрирование
  - Сопровождение
  - Архитектура

Big Data проекты в банках, телекоме и в рознице.

# Правила вебинара



Активно  
участвуем



Off-topic обсуждаем  
в Telegram



Задаем вопрос  
в чат или голосом



Вопросы вижу в чате,  
могу ответить не сразу

## Условные обозначения



Индивидуально



Время, необходимое  
на активность



Пишем в чат



Говорим голосом



Документ



Ответьте себе или  
задайте вопрос

# Карта курса



# Маршрут вебинара

Консенсус

Zookeeper

Kraft

Брокеры

Рефлексия



# Цели вебинара

1. Научимся устанавливать и настраивать Kafka с Zookeeper
2. Научимся устанавливать и настраивать Kafka с Kraft
3. Научимся настраивать брокеры

# Смысл

1. Сможем устанавливать Kafka
2. Сможем настраивать брокеры

# Apache Kafka

# Что такое Kafka

**Kafka** - это *распределенная потоковая платформа*, которая обладает тремя основными возможностями:

- публиковать записи и подписываться на очереди сообщений
- хранить записи с отказоустойчивостью
- обрабатывать потоки по мере их возникновения

# Консенсус

# Распределённый консенсус

- Алгоритмы достижения **консенсуса** в распределённых системах позволяют нескольким процессам достичь **согласия относительно некоторого значения**
- Процессы должны согласовать некоторое значение, предложенное одним из участников, даже если некоторые из них дадут сбой
- Консенсус нужен для расположения событий в определённом порядке и обеспечения согласованности между участниками
- Процесс считается *корректным*, если он не дал сбой и продолжает выполнение шагов алгоритма

# Свойства алгоритмов достижения консенсуса

- Согласованность (*agreement*)

Значение, по которому принимается решение, одинаково для всех корректных процессов

- Действительность (*validity*)

Принятое значение было предложено одним из процессов

- Завершаемость (*termination*)

Все корректные процессы в конечном итоге приходят к решению



# Алгоритмы достижения консенсуса

- Рассылка
- Атомарная рассылка
- Паксос
- Raft
- Византийский консенсус

# Протокол атомарной рассылки Zookeeper

# ZAB - Zookeeper Atomic Broadcast

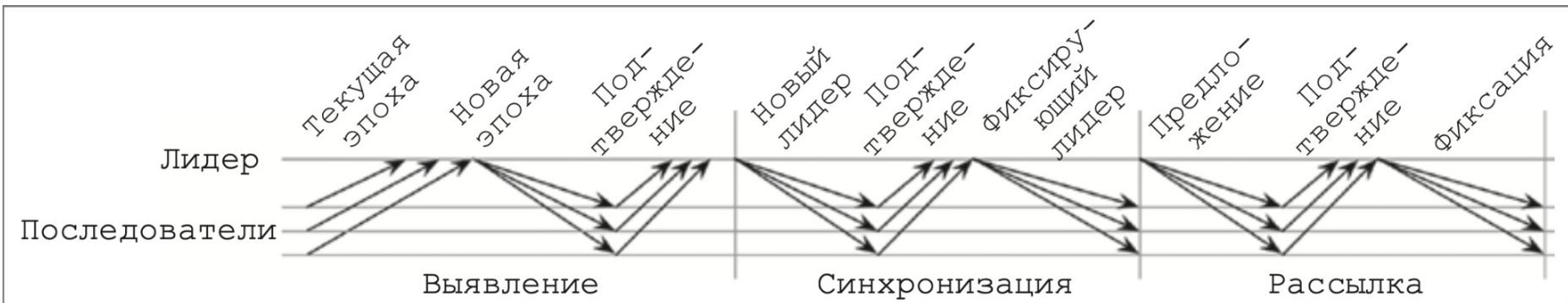
- Процессы могут играть одну из ролей: *лидер* (leader) и *последователь* (follower)
- Лидер:
  - управляет выполнением шагов алгоритма
  - рассыпает сообщения последователям
  - устанавливает порядок событий
- Временная шкала протокола разбита на эпохи
- Процесс начинается с поиска *потенциального лидера*



# Протокол

- Выявление (discovery)
  - Потенциальный лидер предлагает новую эпоху
  - Последователи отвечают идентификатором последней транзакции
- Синхронизация (synchronisation)
  - Потенциальный лидер предлагает себя на роль лидера
  - Роль лидера утверждается после получения подтверждений
- Рассылка (broadcast)
  - Активный обмен сообщениями

# Общая схема протокола



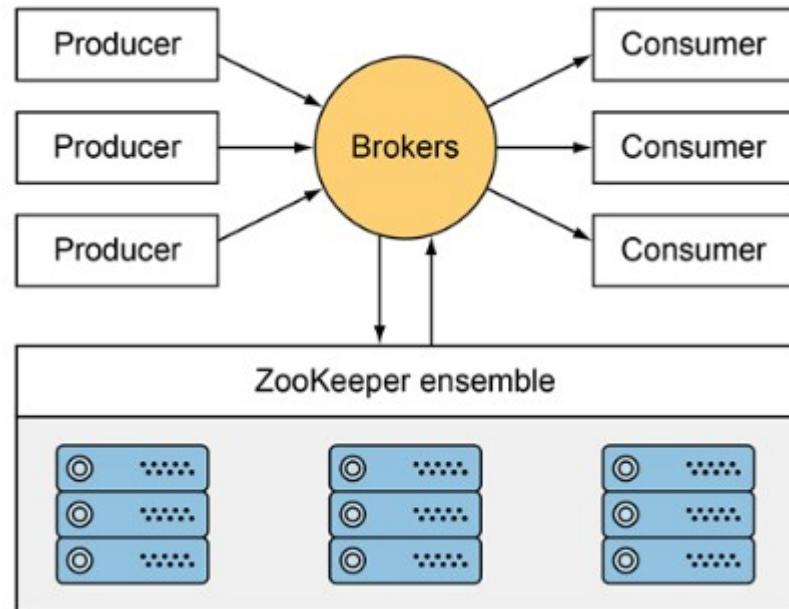
# Kafka с Zookeeper

# Архитектура

**Broker** - управление данными, взаимодействие с клиентами

**Zookeeper** - членство брокеров в кластере, выборы контроллера

**Контроллер** - это брокер, который отвечает за выбор ведущих реплик для разделов



# Настраиваем Zookeeper

- Настраиваем config/zookeeper.properties:

```
dataDir=/opt/kafka/data/zookeeper # каталог, в котором хранится моментальный снимок
clientPort=2181 # порт для подключений клиентов
maxClientCnxns=0 # количество одновременных подключений одного клиента
tickTime=2000 # длина одного такта (измеряется в миллисекундах)
initLimit=20 # кол-во тактов для подключения к лидеру
syncLimit=5 # кол-во тактов, необходимое для синхронизации
server.1=kafka1:2888:3888 # серверы ансамбля
server.2=kafka2:2888:3888 # server.X= имя_хоста:одноранговый_порт:ведущий_порт
server.3=kafka3:2888:3888 # одноранговый_порт – порт для взаимодействия друг с другом
# ведущий_порт – порт, через который выбирается ведущий узел
```

- Создаём data/zookeeper/myid

# Запускаем Zookeeper

- Запускаем:
  - bin/zookeeper-server-start.sh -daemon config/zookeeper.properties
- Проверяем:
  - bin/zookeeper-shell.sh localhost:2181

```
Connecting to localhost:2181
```

```
Welcome to ZooKeeper!
```

```
JLine support is disabled
```

```
WATCHER::
```

```
WatchedEvent state:SyncConnected type:None path:null
```

```
config
```

```
server.1=kafka1:2888:3888:participant
```

```
server.2=kafka2:2888:3888:participant
```

```
server.3=kafka3:2888:3888:participant
```

```
version=0
```

```
ls /
```

```
[zookeeper]
```



# Настраиваем Zookeeper как сервис

- Создаём /usr/lib/systemd/system/zookeeper.service:

[Unit]

Description=ZooKeeper Service

Documentation=https://zookeeper.apache.org/

Requires=network.target

After=network.target

[Service]

Type=simple

User=kafka

Group=kafka

Environment=JAVA\_HOME=/usr/lib/jvm/java-11

ExecStart=/opt/kafka/bin/zookeeper-server-start.sh /opt/kafka/config/zookeeper.properties

ExecStop=/opt/kafka/bin/zookeeper-server-stop.sh

[Install]

WantedBy=multi-user.target

# Запускаем Zookeeper как сервис

- Запускаем:

- sudo systemctl daemon-reload
  - sudo systemctl start zookeeper

- Проверяем:

- sudo systemctl status zookeeper

- `zookeeper.service - ZooKeeper Service`

```
Loaded: loaded (/usr/lib/systemd/system/zookeeper.service; disabled; vendor preset:  
disabled)
```

```
Active: active (running) since Вс 2023-05-14 20:06:58 MSK; 1min 23s ago
```

```
Docs: https://zookeeper.apache.org/
```

```
Main PID: 5415 (java)
```

```
CGroup: /system.slice/zookeeper.service
```

```
└─5415 /usr/lib/jvm/java-11/bin/java -Xmx512M -Xms512M -server -XX:+UseG1GC -
```

```
XX:MaxGCPauseMillis=20 -XX:InitiatingHeapOccupancyPercent=35 -XX:+ExplicitGCInvokesConcurrent -  
XX:MaxInlineLevel=15 -Djava.awt.headless=true...
```

# Настраиваем брокер для Zookeeper

- Настраиваем config/server.properties:

```
broker.id=1
listeners=PLAINTEXT://0.0.0.0:9092
advertised.listeners=PLAINTEXT://kafka1:9092
log.dirs=/opt/kafka/data/kafka-logs
zookeeper.connect=kafka1:2181,kafka2:2181,kafka3:2181
zookeeper.connection.timeout.ms=18000
```

- Запускаем брокер:

- bin/kafka-server-start.sh -daemon config/server.properties

- Проверяем:

- bin/kafka-cluster.sh cluster-id --bootstrap-server kafka1:9092,kafka2:9092,kafka3:9092

Cluster ID: 5K05EPQRRCTGDKcta3cmQ

# Настраиваем брокер как сервис

- Создаём /usr/lib/systemd/system/kafka.service:

```
[Unit]
Description=Kafka Service
Documentation=https://kafka.apache.org/
Requires=network.target
After=network.target zookeeper.service

[Service]
Type=simple
User=kafka
Group=kafka
WorkingDirectory=/opt/kafka
Environment=JAVA_HOME=/usr/lib/jvm/java-11
ExecStart=/opt/kafka/bin/kafka-server-start.sh /opt/kafka/config/server.properties
ExecStop=/opt/kafka/bin/kafka-server-stop.sh

[Install]
WantedBy=multi-user.target
```

# Запускаем брокер как сервис

- Запускаем:

- sudo systemctl daemon-reload
  - sudo systemctl start kafka

- Проверяем:

- sudo systemctl status kafka

- kafka.service - Kafka Service

```
Loaded: loaded (/usr/lib/systemd/system/kafka.service; disabled; vendor preset: disabled)
Active: active (running) since Вс 2023-05-14 20:34:43 MSK; 25s ago
          Docs: https://kafka.apache.org/
Main PID: 8241 (java)
```

```
CGroup: /system.slice/kafka.service
         └─8241 /usr/lib/jvm/java-11/bin/java -Xmx1G -Xms1G -server -XX:+UseG1GC -
```

```
XX:MaxGCPauseMillis=20 -XX:InitiatingHeapOccupancyPercent=35 -XX:+ExplicitGCInvokesConcurrent -
XX:MaxInlineLevel=15 -Djava.awt.headless=true -Xl...
```



# Что не так с Zookeeper?

- Zookeeper - отдельная распределённая система
- Обновления метаданных записываются в Zookeeper синхронно, но отправляются брокерам асинхронно
- При каждом перезапуске контроллер должен прочитать все метаданные из Zookeeper, а затем отправить эти метаданные брокерам
- Некоторые операции выполняются через контроллер, другие - через брокер, а третьи - на Zookeeper

# Raft

# Raft

- Появился в 2013
- Легкий для понимания и реализации
- Используется в CockroachDB, Etcd, Consul, RabbitMQ

# Raft

- Участники хранят локально журнал, содержащий последовательность команд, выполняемых конечным автоматом
- Лидер используется для координации репликации и манипуляций конечного автомата
- Лидер избирается на промежуток времени, называемый *периодом*

# Роли участников

- *Кандидат (candidate)*

Чтобы стать лидером надо набрать большинство голосов

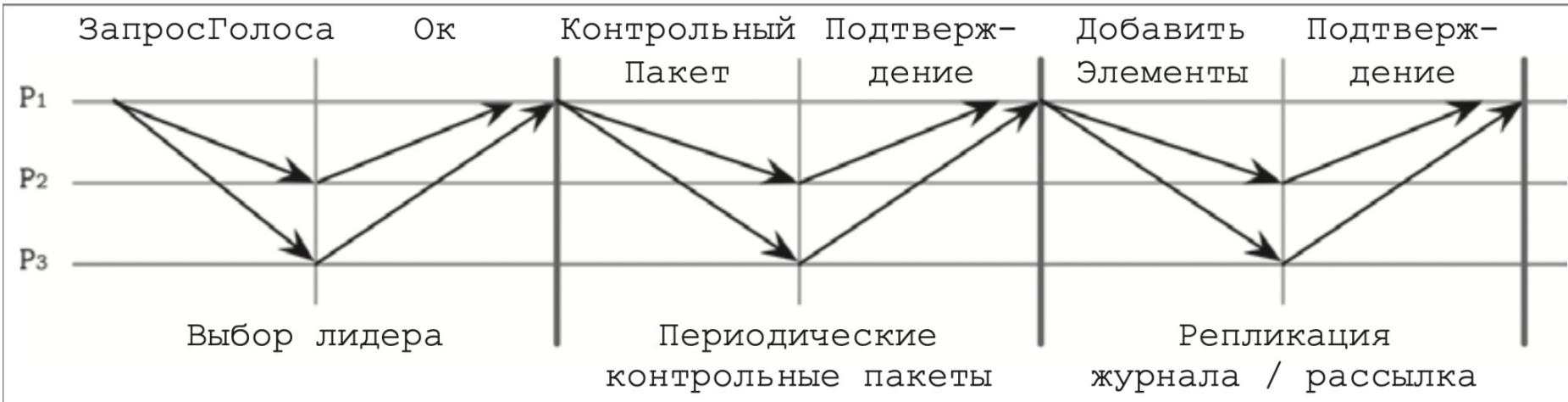
- *Лидер (leader)*

Обрабатывает клиентские запросы и взаимодействует с реплицируемым конечным автоматом (добавляет записи, рассыпает)

- *Последователь (follower)*

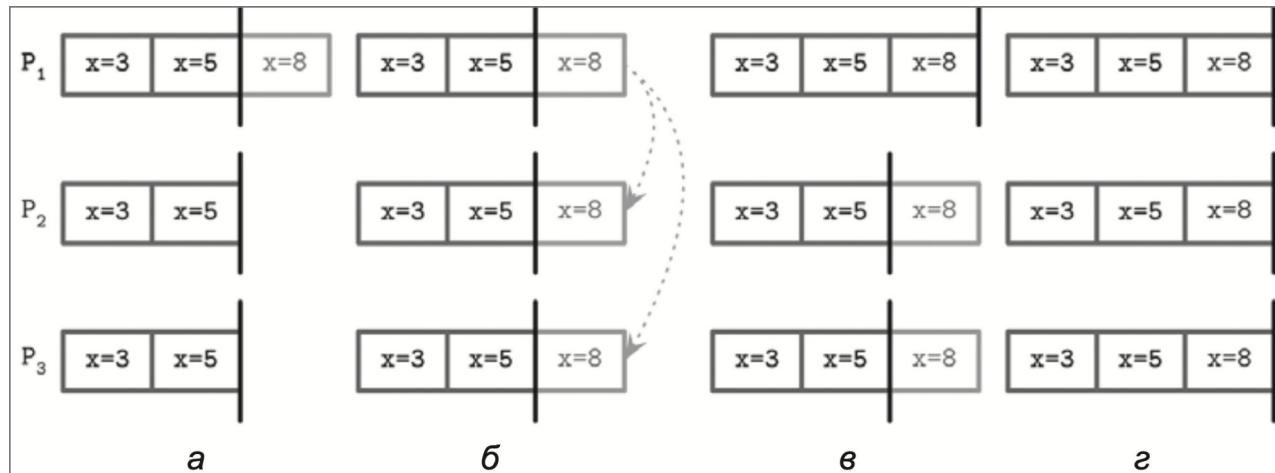
Пассивный участник, сохраняет записи журнала и отвечает на запросы от лидера и кандидатов

# Общая схема алгоритма



# Процедура фиксации

- Новая команда добавляется в журнал лидера
- Новое значение реплицируется на большинство участников
- После завершения репликации лидер фиксирует значение локально
- Решение о фиксации реплицируется на последователей



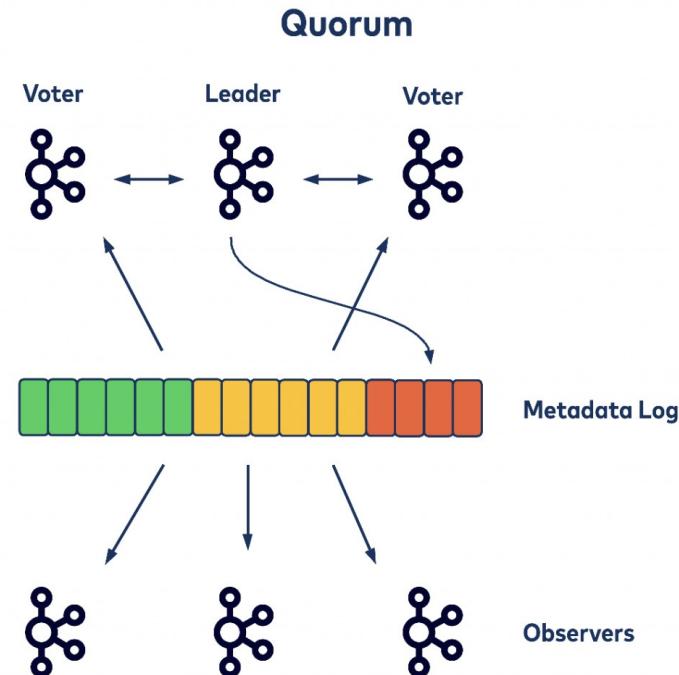
# Гарантии алгоритма Raft

- Только один лидер может быть избран одновременно на заданный период
- Лидер только добавляет новые сообщения в журнал (не удаляет, не переупорядочивает)
- Зафиксированные записи присутствуют в журналах последующих лидеров, журнал нельзя вернуть в прежнее состояние
- Все сообщения однозначно идентифицируются по идентификаторам сообщений и периодов

# Kafka с KRaft

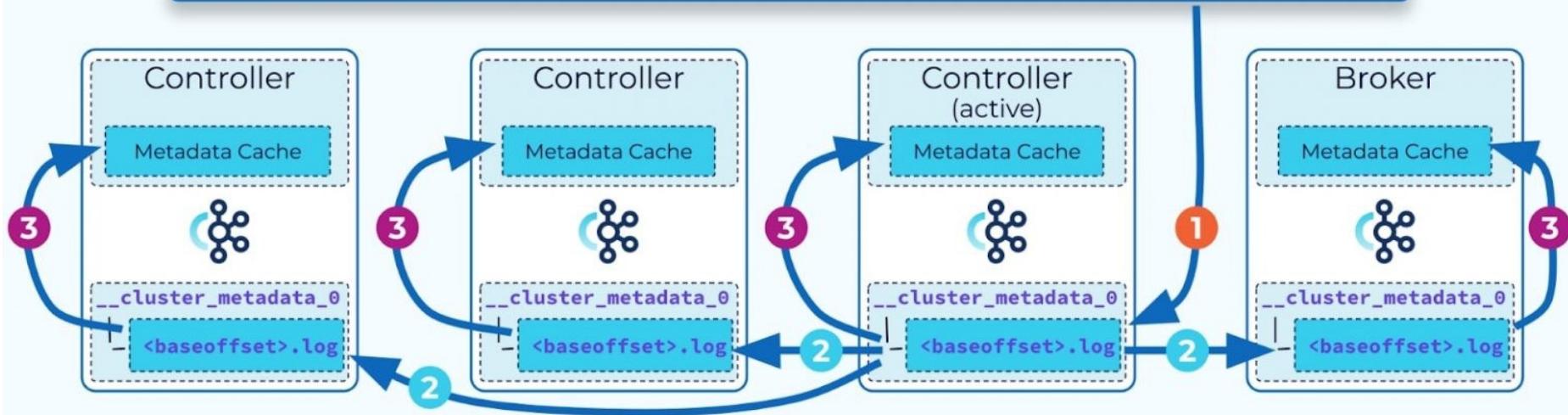
# Архитектура

- Kafka сервер может быть:
  - broker
  - controller (3 или 5)
  - broker, controller (для разработки)
- Узлы контроллера - кворум Raft
- Журнал метаданных - информация о каждом изменении метаданных кластера
- Активный контроллер - лидер Raft



# Управление метаданными

1 Active controller writes the metadata change records to the `__cluster_metadata` topic



2 Controllers (followers) and brokers (observers) replicate `__cluster_metadata` topic events.

3 Controllers and brokers update metadata cache from committed log records

# Настраиваем брокер для KRaft

- Настраиваем config/kraft/server.properties:

```
process.roles=broker,controller
node.id=1
controller.quorum.voters=1@kafka1:9093,2@kafka2:9093,3@kafka3:9093
listeners=PLAINTEXT://0.0.0.0:9092,CONTROLLER://kafka1:9093
inter.broker.listener.name=PLAINTEXT
advertised.listeners=PLAINTEXT://kafka1:9092
controller.listener.names=CONTROLLER
listener.security.protocol.map=CONTROLLER:PLAINTEXT,PLAINTEXT:PLAINTEXT,SSL:SSL,SASL_PLAINTEXT:
SASL_PLAINTEXT,SASL_SSL:SASL_SSL
log.dirs=/opt/kafka/data/kraft-combined-logs
```

- Создаём CLISTER ID (на одном узле):

- `export KAFKA_CLUSTER_ID="$(bin/kafka-storage.sh random-uuid)"`

- Форматируем log каталог (на каждом узле):

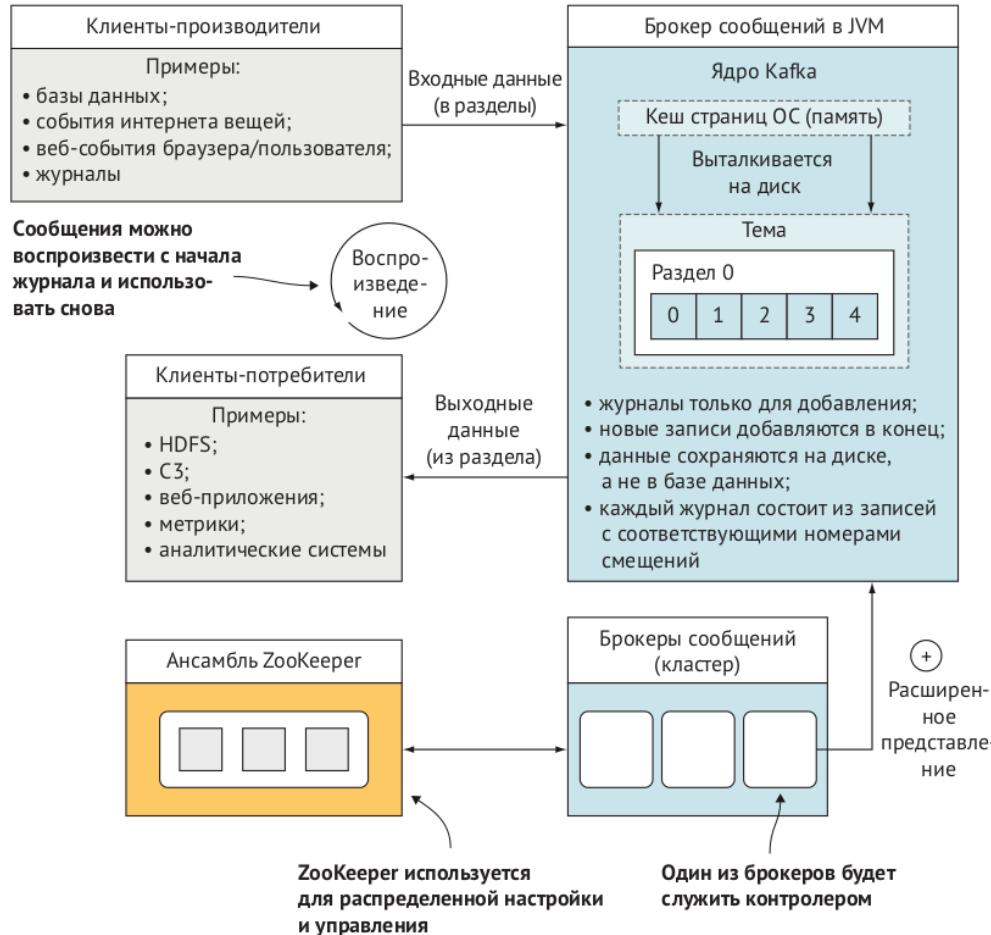
- `bin/kafka-storage.sh format -t $KAFKA_CLUSTER_ID -c config/kraft/server.properties`

# Запускаем брокер с KRaft

- Запускаем брокер:
  - `bin/kafka-server-start.sh -daemon config/kraft/server.properties`
- Проверяем:
  - `bin/kafka-cluster.sh cluster-id --bootstrap-server kafka1:9092,kafka2:9092,kafka3:9092`  
Cluster ID: x0NST8QuSdGKIhupydw63Q
  - `bin/kafka-metadata-shell.sh --snapshot /opt/kafka/data/kraft-combined-logs/bootstrap.checkpoint`

# Брокеры

# Брокер



# Основные параметры

- broker\_id, node\_id
- listeners, listener.security.protocol.map
- zookeeper.connect, controller.quorum.voters
- log.dirs
- num.recovery.threads.per.data.dir – кол-во потоков на каждый каталог журналов
- auto.create.topics.enable
- auto.leader.rebalance.enable – запуск перебалансировки через leader.imbalance.check.interval.seconds, если дисбаланс ведущих реплик превышает leader.imbalance.per.broker.percentage
- delete.topic.enable



# Параметры топиков по умолчанию

- num.partitions
- default.replication.factor
- log.retention.ms, log.retention.minutes – промежуток времени, по истечении которого сообщения удаляются
- log.retention.bytes – объём сохраняемых сообщений в одном разеделе
- log.segment.bytes – размер активного сегмента
- log.roll.ms – отрезок времени, по истечении которого сегмент закрывается
- min.insync.replicas – минимальное кол-во реплик для подтверждения успешной записи
- message.max.bytes – максимальный размер сообщений



# Вопросы?



Ставим “+”,  
если вопросы есть



Ставим “-”,  
если вопросов нет

# Рефлексия

# Рефлексия



С какими впечатлениями уходите с вебинара?



Как будете применять на практике то,  
что узнали на вебинаре?

# Следующий вебинар



17 мая 2023

## Операции с топиками



Ссылка на вебинар  
будет в ЛК за 15 минут



Материалы  
к занятию в ЛК –  
можно изучать



Обязательный материал  
обозначен красной  
лентой

**Заполните, пожалуйста,  
опрос о занятии  
по ссылке в чате**

Спасибо за внимание!

# Приходите на следующие вебинары



**Заигрин Вадим**

Ведущий эксперт по технологиям, Сбербанк

vzaigrin@yandex.ru  
<https://t.me/vzaigrin>