ADS

Arenadata Streaming

Arenadata Streaming (ADS) – масштабируемая отказоустойчивая система для потоковой обработки данных в режиме реального времени, адаптированная для корпоративного использования и построенная на базе Apache Kafka и Apache NiFi.

Версия CURRENT Язык: RU

Содержание

1. Установка
- Online-установка
  - Установка ADCM
  - Подготовка хостов
  - Установка кластера ADS
    - Создание кластера
    - Добавление сервисов
    - Добавление хостов в кластер
    - Добавление компонентов
    - Настройка сервисов
    - Настройка кластера
    - Установка кластера
  - Установка мониторинга
    - Способ 1. Сервис мониторинга
    - Способ 2. Кластер мониторинга
      - Создание кластера
      - Добавление сервисов
      - Добавление хостов в кластер
      - Добавление компонентов
      - Настройка сервисов
      - Установка кластера
      - Интеграция с кластером ADS
- Offline-установка
  - Установка ADCM
  - Подготовка хостов
  - Установка кластера Enterprise Tools
    - Создание кластера
    - Добавление сервисов
    - Добавление хостов в кластер
    - Добавление компонентов
    - Установка кластера
  - Установка кластера ADS
    - Создание кластера
    - Добавление сервисов
    - Добавление хостов в кластер
    - Добавление компонентов
    - Настройка сервисов
    - Настройка кластера
    - Импорт настроек ET
    - Установка кластера
  - Установка мониторинга
    - Способ 1. Сервис мониторинга
    - Способ 2. Кластер мониторинга
      - Настройка кластера Enterprise Tools
      - Интеграция с кластером ADS

2. Базовые операции
- Kafka
  - Начало работы
  - Управление доступом
    - Аутентификация
      - SASL PLAINTEXT
      - Kerberos
        - MIT Kerberos
        - MS Active Directory
      - Защита канала по протоколу SSL
    - Авторизация
      - Плагин Ranger Kafka
      - Использование Ranger в Kafka
      - ACL в Kafka
  - Примеры использования
    - Создание приложения Kafka Streams
    - Использование API
    - Cruise Control
  - Установка в режиме Kraft
  - Миграция Kafka на Kraft
  - Настройка и использование Tiered Storage
- Kafka Connect
  - Mirror Maker 2
- Kafka REST Proxy
  - Пример использования
  - Использование kafka-rest-security
- ksqlDB
  - Пример использования
- NiFi
  - Обзор UI
    - NiFi Server
    - NiFi Registry
  - Управление доступом
    - Аутентификация
    - Авторизация
      - Пользовательский интерфейс
      - ADPS
        - Плагин Ranger NiFi
        - Использование Ranger в NiFi
  - Основные операции
    - Работа с процессорами
      - Настройка процессоров и групп процессов
    - Работа с атрибутами
    - Работа с шаблонами
    - Встроенный мониторинг
  - Использование языка выражений
  - Шифрование параметров конфигурации
  - Примеры использования
    - Использование REST API
    - Построение потока данных
    - NiFi ADB Connector
      - Обзор
      - Пример использования
- Schema Registry
  - Пример использования

3. Руководства
- Управление кластером ADS
  - Действия в кластере
  - Действия в сервисе
    - Kafka
    - Kafka Connect
    - Kafka REST Proxy
    - ksqlDB
    - NiFi
    - NiFi2
    - MiNiFi
    - Schema Registry
    - ZooKeeper
    - Monitoring Clients
    - Monitoring
- Управление доступом
  - Аутентификация
    - Basic
    - Kerberos
      - MIT Kerberos
      - MS Active Directory
      - Samba
      - Custom kerberization settings
    - SASL PLAINTEXT
    - Защита канала по протоколу SSL
  - Авторизация
    - ADPS
      - Обзор
      - Интеграция с кластером ADS
- Мониторинг
  - Метрики

4. Справочные материалы
- Конфигурационные параметры ADS
- Kafka
  - Клиенты
  - API
- NiFi
  - API

5. Подготовка окружения
- Аппаратные требования к серверам
- Аппаратные требования к сети
- Программные требования
- Рекомендации по настройке
  - Kafka
  - NiFi

6. Концепции
- Обзор ADS
  - Сценарии использования
- Архитектура
  - Обзор
  - Kafka
    - Концепции хранения
    - Сжатие журналов
    - Гарантии доставки сообщений
    - Репликация
    - Обзор Kraft
    - Обзор Tiered Storage
    - KTLS
    - Обзор Cruise Control
  - Kafka Connect
    - Обзор
    - Обзор Debezium
  - Kafka REST Proxy
    - Обзор
  - ksqlDB
    - Обзор
  - MiNiFi
    - Обзор
  - NiFi
    - Обзор
  - Schema Registry
    - Обзор
  - ZooKeeper
    - Обнаружение сервисов
    - Выбор лидера
- Модель данных
  - Kafka
    - Формат сообщений
  - NiFi
    - Объекты
    - Типы процессоров

7. Релизы
- Релизы ADS
- Поддерживаемые сервисы ADS
- Матрица совместимости версий

Установка Arenadata Streaming

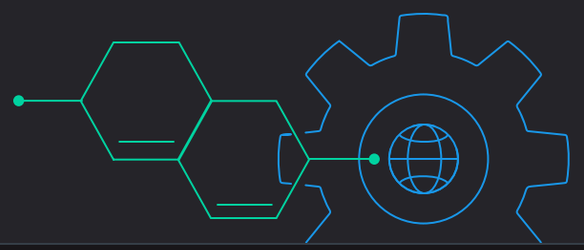
Назад к содержанию

Этот раздел содержит описание установки кластера Arenadata Streaming (ADS). Для установки рекомендуется использовать Arenadata Cluster Manager (ADCM) — отдельный продукт Arenadata, предназначенный для простого, удобного и быстрого развертывания и дальнейшей эксплуатации программного обеспечения.

Возможны два способа установки ADS через ADCM.

Online-установка

на хостах с доступом к Интернету



Offline-установка

на хостах без доступа к Интернету

Online-установка

Назад к содержанию

Этот раздел посвящен **online**-установке ADS. Данный способ установки подходит для хостов с доступом к Интернету.

Процесс online-установки через ADCM включает следующие шаги.

Шаг 1.

Установка ADCM

Шаг 2.

Подготовка хостов

Шаг 3.

Установка кластера ADS

Шаг 4.

Установка мониторинга

Установка ASCM

Содержание

Часть 1. Предварительные требования
Требования к аппаратному обеспечению
Требования к системе
Установка Docker
Установка Docker Compose
Часть 2. Установка ASCM
Часть 3. Запуск ASCM
Часть 4. Проверка установки
Проблемы запуска Docker-контейнеров
Часть 5. Обновление ASCM

Перед установкой ASCM убедитесь, что ваша система соответствует минимальным требованиям, описанным в этом разделе.

Часть 1. Предварительные требования

Требования к аппаратному обеспечению

Для установки и работы ASCM требуется следующее оборудование (Hardware):

- Архитектура: x86_64 (AMD64)
- CPU: 4 ядра
- RAM: 8 ГБ
- Диск: 40 ГБ
- Архитектура: x86_64 (AMD64)
- CPU: 4 ядра
- RAM: 8 ГБ
- Диск: 40 ГБ (рекомендуется использовать SSD)

Требования к системе

Для установки и работы ASCM требуется следующее программное обеспечение (Software):

- Операционная система: Ubuntu 22.04 LTS
- Docker: версия 20.10.0 или выше
- Docker Compose: версия 2.0.0 или выше

Docker и Docker Compose должны быть установлены заранее. Если они отсутствуют, установите их, следуя инструкциям в разделах Установка Docker и Установка Docker Compose.

Установка Docker

Docker — это платформа для разработки, доставки и запуска приложений в контейнерах. Docker позволяет упаковать приложение со всеми его зависимостями в стандартный контейнер, который можно запустить на любой системе с поддержкой Docker.

Docker Compose — это инструмент для определения и запуска многоконтейнерных приложений Docker. Docker Compose использует файл docker-compose.yml для настройки сервисов приложения и позволяет запускать все сервисы одной командой.

Docker и Docker Compose должны быть установлены заранее. Если они отсутствуют, установите их, следуя инструкциям в разделах Установка Docker и Установка Docker Compose.

Docker — это платформа для разработки, доставки и запуска приложений в контейнерах. Docker позволяет упаковать приложение со всеми его зависимостями в стандартный контейнер, который можно запустить на любой системе с поддержкой Docker

Подготовка хостов

Назад к содержанию

Перед установкой нового кластера необходимо добавить информацию о его будущих хостах в ADCM. Это задача **хостпровайдеров** — специальных плагинов, отвечающих за взаимодействие между хостами и ADCM.

ADCM поддерживает несколько типов хостпровайдеров. Для получения подробной информации о том, как их настроить и использовать для создания хостов, обратитесь к документации:

- SSH
- Yandex Cloud
- VCloud (VMware)
- Cloud.ru Advanced
- CROC Cloud

ВНИМАНИЕ

Если ADS или ADS Control планируется использовать совместно с ADPS, при создании хостов ADS/ADS Control необходимо указывать их полные доменные имена (FQDN). Максимальная длина FQDN — 48 символов.

Создание кластера

Ольга Семме, Дарья Барышева

Содержание

Назад к содержанию

Шаг 1. Загрузка бандла

Шаг 2. Загрузка бандла в ADCM

Шаг 3. Создание кластера на базе загруженного бандла

В ADCM **кластер** означает набор сервисов, запущенных на группе хостов.

Дистрибутивы кластеров для ADCM поставляются в виде бандлов. В общем случае бандл представляет обычный архив, описывающий логику развертывания кластера.

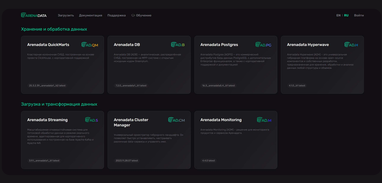
Шаги для добавления нового кластера приведены ниже.

ПРИМЕЧАНИЕ

При использовании Enterprise-версии продукта следует пропустить шаг 1, получить бандл у команды поддержки Arenadata и перейти к шагу 2.

Шаг 1. Загрузка бандла

1. Перейдите на сайт https://network.arenadata.io/ и выберите продукт ***Arenadata Streaming***.



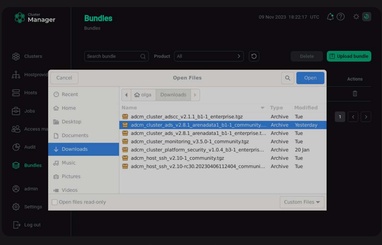
Переход к загрузке дистрибутивов

2. На следующей странице загрузите бандл для нужной версии ADS.

Шаг 2. Загрузка бандла в ADCM

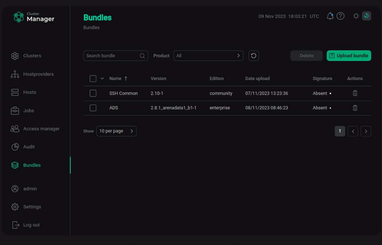
Чтобы загрузить бандл в ADCM:

1. Выберите пункт ***Bundles*** в левом навигационном меню и нажмите ***Upload bundle***.
2. Выберите бандл в открывшемся диалоговом окне.



Переход к загрузке бандла

3. После успешной загрузки бандл отображается на странице ***Bundles***.



Результат успешной загрузки бандла

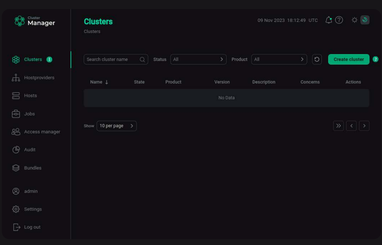
Шаг 3. Создание кластера на базе загруженного бандла

ВАЖНО

На этапе создания кластера через web-интерфейс ADCM генерируется только сам кластер. Это означает добавление записи о кластере в ADCM — сервисы на этом шаге еще не устанавливаются.

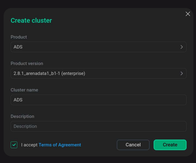
Для создания кластера необходимо выполнить шаги:

1. Выберите пункт ***Clusters*** в левом навигационном меню и нажмите ***Create cluster***.



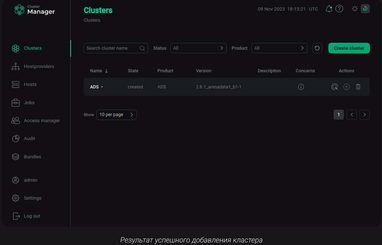
Переход к созданию кластера

2. В открывшемся окне требуется описать новый кластер:
   - Выберите загруженный бандл в поле ***Product***.
   - Выберите версию бандла в поле ***Product version***. Несколько версий становятся доступны после загрузки нескольких бандлов для одного типа продуктов.
   - Введите имя кластера в поле ***Cluster name***.
   - При необходимости добавьте описание кластера в поле ***Description***.
   - Установите флаг ***I accept Terms of Agreement*** для подписания лицензионного соглашения. Флаг отображается только для бандлов версии Enterprise. Чтобы ознакомиться с текстом соглашения, перейдите по ссылке ***Terms of Agreement***. Подписать соглашение также возможно на странице ***Bundles***, нажав на соответствующий продуктовый бандл.
   - Нажмите ***Create***.



Создание кластера

3. После успешного добавления кластер отображается на странице ***Clusters***.



Результат успешного добавления кластера

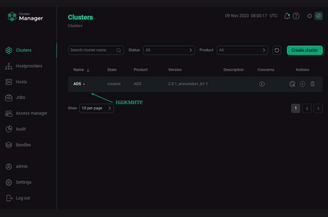
Добавление сервисов

Ольга Семак, Дарья Барышева

Назад к содержанию

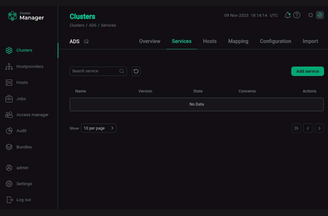
В ADCM **сервис** означает программное обеспечение, выполняющее некоторую функцию. Примеры сервисов в кластерах ADS: NiFi, Kafka, ZooKeeper и другие. Для добавления сервисов в кластер:

1. Выберите кластер на странице **Clusters**. Для этого нажмите на имя кластера в столбце **Name**.



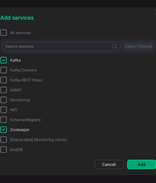
Выбор кластера

2. Откройте вкладку **Services** на странице кластера и нажмите ***Add service***.



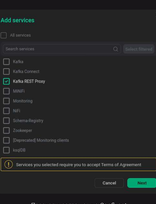
Переход к добавлению сервисов

3. В открывшемся окне выберите сервисы для добавления в кластер и нажмите ***Add***.



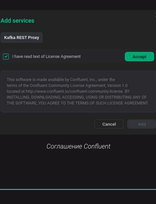
Выбор сервисов

При добавлении сервисов ksqlDB, Schema-Registry, Kafka REST Proxy требуется подписать лицензионное соглашение Confluent. Для этого перейдите к соглашению, нажав ***Next***.



Переход к соглашению Confluent

Далее ознакомьтесь с текстом соглашения и подпишите его, нажав ***Accept***.



Соглашение Confluent

Нажмите ***Add*** для добавления сервиса.

Краткое описание доступных сервисов приведено ниже.

Сервисы, которые могут быть добавлены в кластер ADS

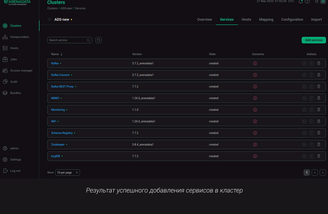
Сервис	Назначение
ZooKeeper	Сервис централизованной координации распределенных приложений, используется для хранения метаданных о разделах (partitions) топиков (topics) и брокеров Apache Kafka, а также для выбора брокера в качестве контроллера Kafka. Сообщает брокерам Kafka о текущем состоянии кластера
Kafka	Распределенная платформа потоковой передачи событий с открытым исходным кодом, используемая для высокопроизводительных конвейеров данных, потоковой аналитики, интеграции данных и критически важных приложений
Kafka REST Proxy	Сервис предоставляет RESTful-интерфейс для кластера Kafka, что упрощает создание и прием сообщений, просмотр состояния кластера и выполнение административных действий без использования собственного протокола Kafka или клиентов
Schema-registry	Сервис выполняет функции: • Хранение версионной истории всех схем Avro на основе указанной стратегии имен субъектов с поддержкой нескольких параметров совместимости. • Возможность изменения схем Avro в соответствии с настроенными параметрами совместимости и расширенной поддержкой этих типов схем. • Обеспечение интерфейса RESTful для хранения и извлечения схем Avro. • Предоставление сериализаторов, которые подключаются к клиентам Apache Kafka, обрабатывающим хранение и извлечение схемы сообщений.
ksqlDB	Сервис представляет собой потоковый SQL-движок с открытым исходным кодом, который обеспечивает обработку данных в реальном времени с использованием Apache Kafka
NiFi	Распределенная платформа обработки событий (сообщений), обеспечивающая управление потоками данных из различных источников в режиме реального времени с помощью графического интерфейса
NiFi2	Сервис NiFi с версией 2.x. Сервис добавлен в режиме технологического превью и не предназначен для использования в производственных средах. Существующие ограничения для использования NiFi2: • NiFi2 может быть установлен только в составе отдельного кластера (NiFi2 + Zookeeper). • Перед установкой сервиса необходимо включить установку Java из репозиториев arenadata, используя параметр ***arenadata_java*** репозитория используемой ОС для установки Java 21 при настройке кластера.
MiNiFi	Подпроект Apache NiFi, представляющий собой дополнительный подход к сбору данных, который дополняет основные принципы NiFi в управлении потоками данных, фокусируясь на сборе данных в источнике их создания
Monitoring	Сервис разворачивает в ADS свой сервер Prometheus, который реализует сбор и хранение метрик мониторинга кластера ADS, а также поддерживает возможность использования веб-приложения Grafana для визуализации и анализа информации
Monitoring Clients	Агенты, отправляющие информацию о хосте и сервисах в мониторинг. Сервис необходимо добавить после установки и настройки кластера мониторинга, если планируется мониторинг текущего кластера
Kafka Connect	Инструмент для масштабируемой и надежной потоковой передачи данных между Kafka и другими системами данных

Минимальный набор сервисов, рекомендуемый для кластеров ADS, приведен ниже:

- NiFi
- Kafka
- ZooKeeper

Эти сервисы составляют основу Streaming, и их достаточно для быстрой и простой установки и управления потоковой передачей с помощью Arenadata Cluster Manager. Полный перечень сервисов зависит от требований конкретного проекта. Могут быть установлены не все сервисы. Например, если вы не планируете использовать NiFi, то добавлять сервис не требуется. Если использовать сервис мониторинга (не на основе Graphite), нет необходимости устанавливать агенты из клиентов мониторинга. Если вы планируете использовать Kafka, вам потребуются службы Kafka и ZooKeeper. Сервис может состоять из обязательных и необязательных компонентов. Например, сервис ksqlDB состоит из обязательного серверного компонента и необязательного клиентского компонента.

4. В результате успешно добавленные сервисы отобразятся на вкладке **Services**.



Результат успешного добавления сервисов в кластер

ПРИМЕЧАНИЕ

Опциональные сервисы могут быть добавлены в кластер позднее. Процесс добавления сервисов в уже развернутый кластер не отличается от описанного выше.

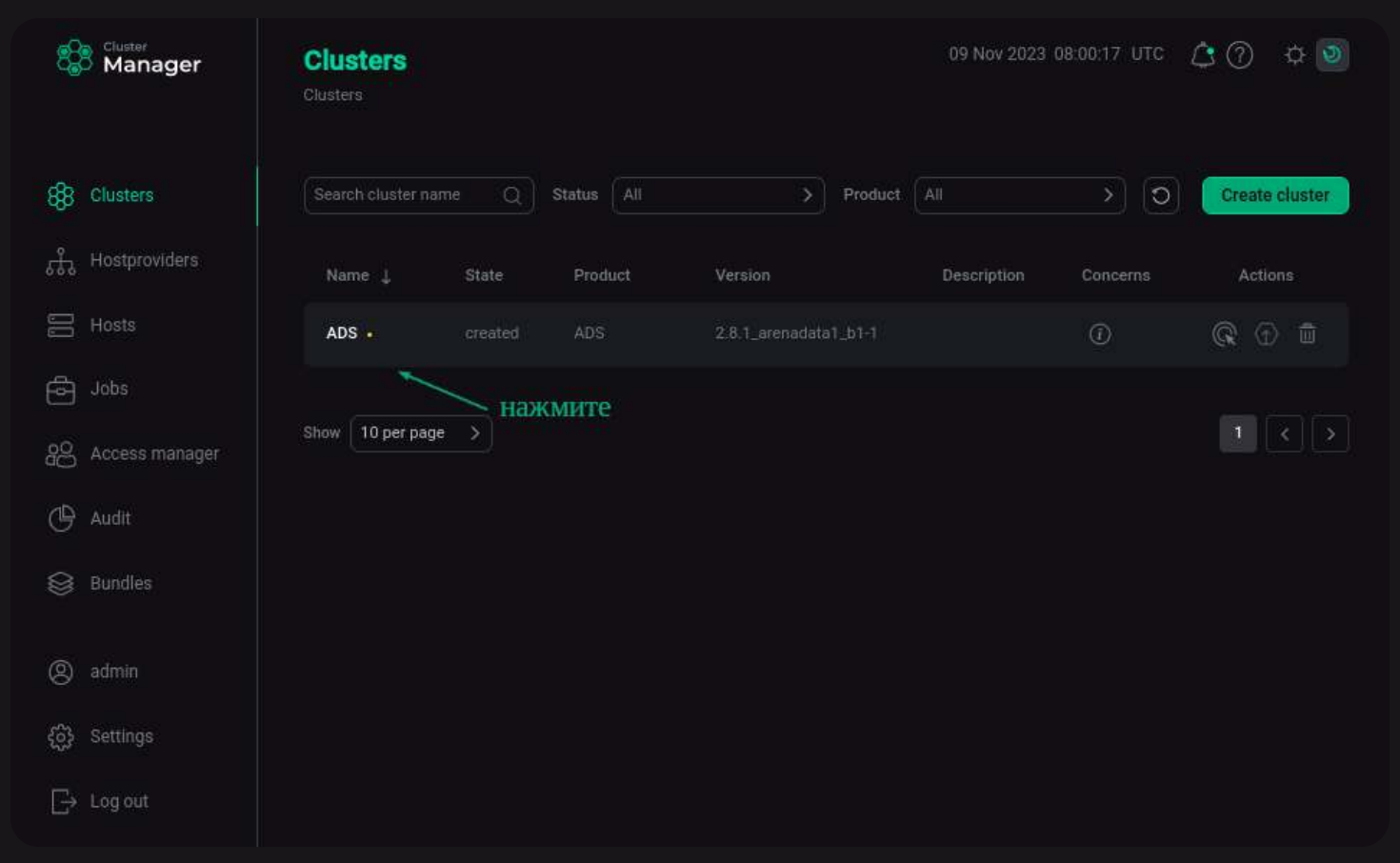
Добавление хостов в кластер

Ольга Семме, Дарья Барышева

Назад к содержанию

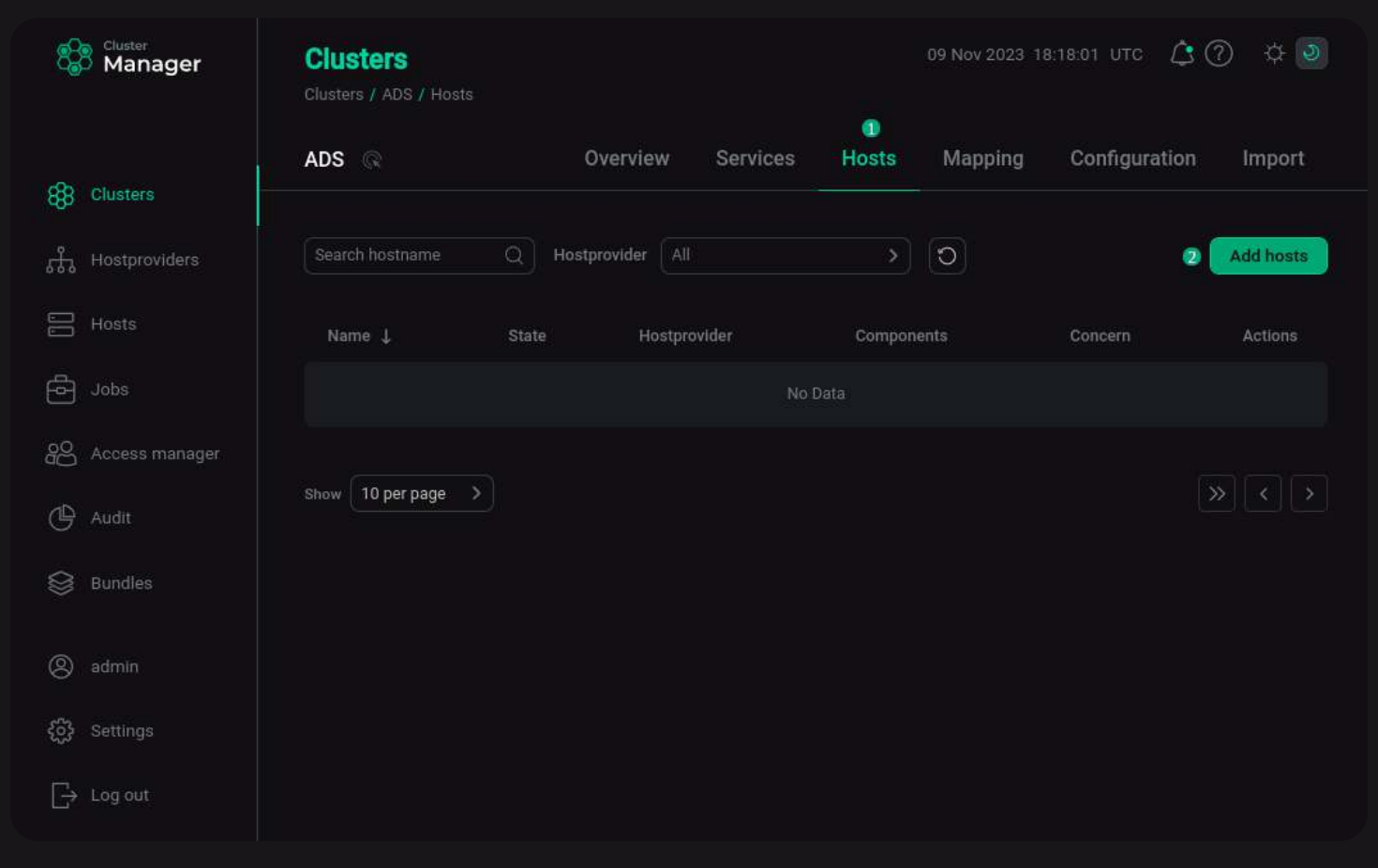
Для добавления хостов в кластер:

1. Выберите кластер на странице ***Clusters***. Для этого нажмите на имя кластера в столбце ***Name***.



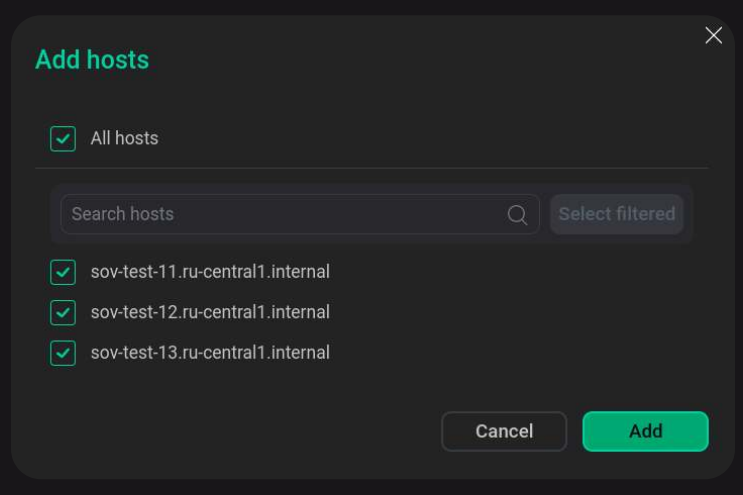
Выбор кластера

2. Откройте вкладку ***Hosts*** на странице кластера и нажмите ***Add hosts***.



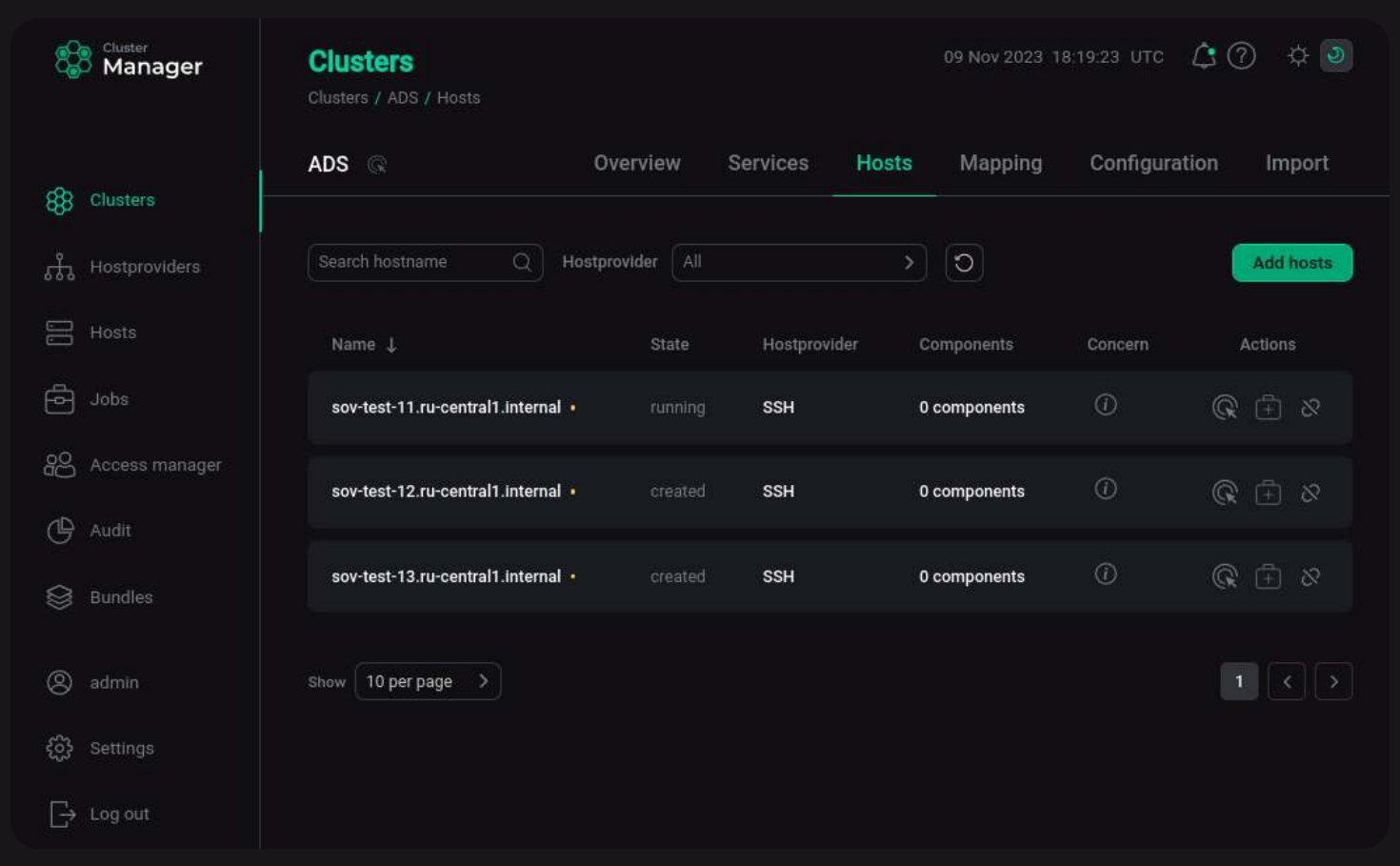
Переход к добавлению хостов

3. В открывшемся окне выберите хосты для добавления в кластер и нажмите ***Add***.



Выбор хостов

4. В результате успешно добавленные хосты отображаются на вкладке ***Hosts***.



Результат успешного добавления хостов в кластер

Добавление компонентов

Ольга Семина, Дарья Барышева

Назад к содержанию

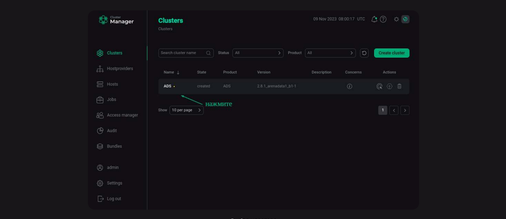
В ADCM **компонент** означает часть сервиса, которая должна быть развернута на одном или нескольких хостах кластера. Например, сервис NiFi состоит из таких компонентов, как NiFi Server и NiFi Registry.

ВАЖНО

Изначально на хостах кластера нет ни одного компонента — распределение компонентов между хостами должно быть проведено вручную.

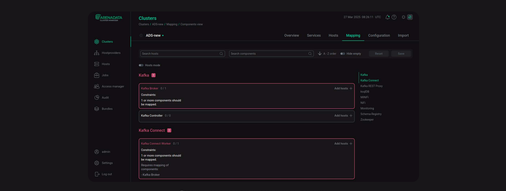
Для добавления компонентов на хосты кластера:

1. Выберите кластер на странице *Clusters*. Для этого нажмите на имя кластера в столбце *Name*.



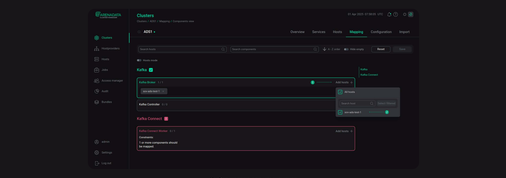
Выбор кластера

2. Откройте вкладку *Mapping* на странице кластера.



Переход к распределению сервисных компонентов

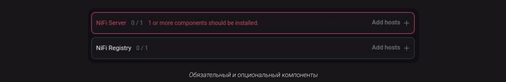
3. Распределите все компоненты между хостами кластера. Для назначения хостов компоненту нажмите *Add hosts* и выберите хосты в открывшемся окне.



Выбор хостов для компонента

Обязательные и опциональные компоненты

Каждый сервис может иметь обязательные и опциональные компоненты. Обязательные подсвечиваются красным цветом. Также в интерфейсе ADCM для них выводится информация о том, в каком количестве необходимо установить каждый компонент.



Обязательный и опциональный компоненты

ВНИМАНИЕ

Без назначения нужного числа хостов обязательным компонентам сохранение общей схемы распределения невозможно.

Следующая таблица содержит примеры распределения компонентов в кластерах ADS, состоящих из 3 и 5 хостов.

Примеры распределения компонентов

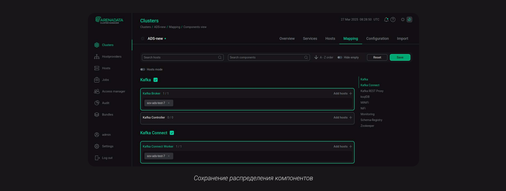
3 хоста

Хост	Сервис	Компонент
Host 1	Zookeeper	Zookeeper server
	Kafka	Kafka Broker
	Kafka REST Proxy	Kafka REST Proxy
	Schema-Registry	Schema-Registry
Host 2	Zookeeper	Zookeeper Server
	ksqlDB	ksqlDB Client ksqlDB Server
	Kafka	Kafka Broker
	NiFi	NiFi Server
Host 3	Zookeeper	Zookeeper Server
	Kafka	Kafka Broker
	Kafka Connect	Kafka Connect Worker
	NiFi	NiFi Server Nifi Registry
	MiNiFi	MiNiFi C2 Server MiNiFi Agent

5 хостов

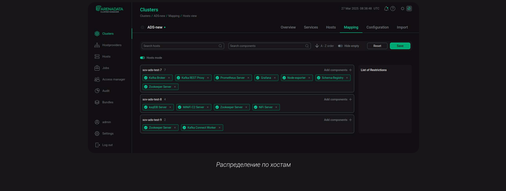
Хост	Сервис	Компонент
Host 1	Zookeeper	Zookeeper Server
	Kafka	Kafka Broker
	Schema-Registry	Schema-Registry
Host 2	Zookeeper	Zookeeper Server
	Kafka	Kafka Broker
Host 3	Zookeeper	Zookeeper Server
	ksqlDB	ksqlDB Client ksqlDB Server
	Kafka REST Proxy	Kafka REST Proxy
Host 4	Kafka	Kafka Broker
	NiFi	NiFi Server NiFi Registry
Host 5	NiFi	NiFi Server
	MiNiFi	MiNiFi C2 Server MiNiFi Agent
	Kafka Connect	Kafka Connect Worker

4. После завершения распределения компонентов нажмите *Save*.



Сохранение распределения компонентов

Начиная с версии 2.3.0 ADCM поддерживает альтернативный способ сопоставления – по хостам. Для этого переведите в активное положение переключатель *Hosts mode* и распределите компоненты между хостами путем нажатия на кнопку *Add components* напротив каждого хоста.



Распределение по хостам

Настройка сервисов

Часть статьи «Дата-платформа»

ADS

Содержание

Назад к содержанию

Конфигурация

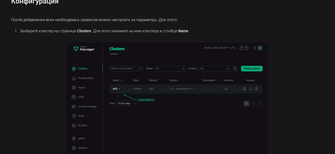
Компоненты

Группы конфигурации

Конфигурация

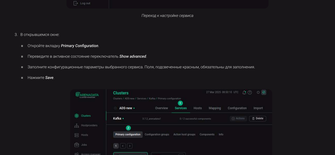
После добавления всех необходимых сервисов можно настроить их параметры. Для этого:

1. Выберите кластер на странице **Clusters**. Для этого нажмите на имя кластера в столбце **Name**



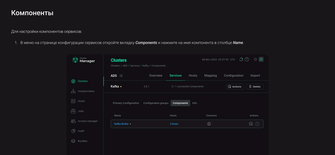
Выбор кластера

2. Откройте вкладку **Services** на странице кластера. Нажмите на имя сервиса в столбце **Name**.

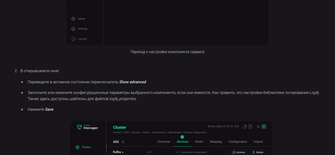


Переход к настройке сервиса

3. В открывшемся окне:
   - Откройте вкладку **Primary Configuration**
   - Переведите в активное состояние переключатель **Show advanced**
   - Заполните конфигурационные параметры выбранного сервиса. Поля, подсвечены красным, обязательны для заполнения
   - Нажмите **Save**



Настройка сервиса



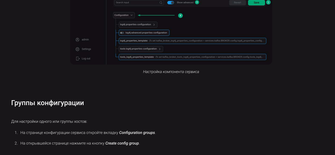
ПРИМЕЧАНИЕ

Более подробную информацию по всем конфигурационным параметрам можно получить в статье Конфигурационные параметры ADS.

Компоненты

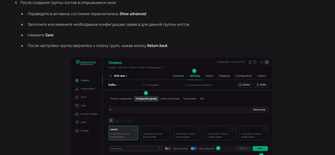
Для настройки компонентов сервисов:

1. В меню на странице кластера сервисов откройте вкладку **Components** и нажмите на имя компонента в столбце **Name**



Переход к настройке компонента сервиса

2. В открывшемся окне:
   - Переведите в активное состояние переключатель **Show advanced**
   - Заполните или измените конфигурационные параметры выбранного компонента, если они имеются. Как правило, это настройки библиотеки log4j2. Также здесь доступны шаблоны для файлов kpdl, properties
   - Нажмите **Save**

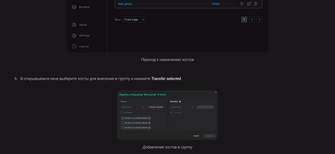


Настройка компонента сервиса

Группы конфигурации

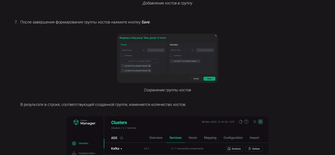
Для настройки одного или группы хостов:

1. На странице конфигурации сервиса откройте вкладку **Configuration groups**
2. На открывшейся странице нажмите на кнопку **Create config group**



Переход к созданию группы хостов

3. В открывшемся окне введите наименование группы и нажмите **Next**



Создание группы хостов

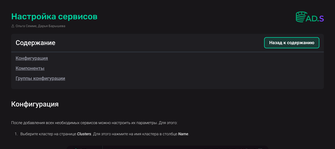
4. После создания группы хостов в открывшемся окне:
   - Переведите в активное состояние переключатель **Show advanced**
   - Заполните или измените необходимые конфигурации сервиса для данной группы хостов
   - Нажмите **Save**
   - После настройки группы вернитесь в списку групп, нажав кнопку **Return back**

Настройка группы хостов

5. Для назначения хостов в группу в старом, соответствующей созданной группе, нажмите на иконку

Переход к назначению хостов

6. В открывшемся окне выберите хосты для внесения в группу и нажмите **Transfer selected**

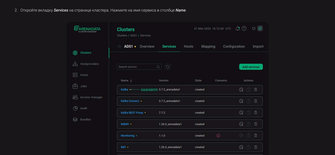


Добавление хостов в группу

7. После завершения формирования группы хостов нажмите кнопку **Save**

Сохранение группы хостов

В результате в группе, соответствующей созданной группе, изменится количество хостов.



Сформированная группа хостов

ПРИМЕЧАНИЕ

Более подробную информацию о настройке сервисов для группы хостов можно получить в статье Настройка конфиг-групп.

Настройка кластера

Ольга Семме, Дарья Барышева

Содержание

Назад к содержанию

Primary configuration

Configuration groups

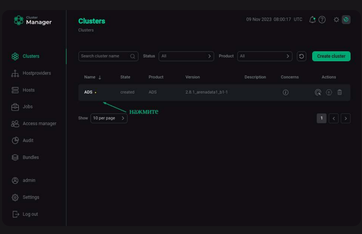
Ansible settings

ВАЖНО

В большинстве случаев кастомная настройка кластера не требуется — значения параметров можно оставить по умолчанию.

После добавления нового кластера можно выполнить его настройку. Для этого выполните шаги:

1. Выберите кластер на странице ***Clusters***. Для этого нажмите на имя кластера в столбце ***Name***.



Выбор кластера

2. Откройте вкладку ***Configuration*** на странице кластера. Вкладка ***Configuration*** состоит из трех подразделов: ***Primary configuration***, ***Configuration groups***, ***Ansible settings***.

3. Заполните необходимые конфигурационные параметры на нужной вкладке и нажмите ***Save***.

Primary configuration

Вкладка ***Configuration → Primary configuration*** содержит основные конфигурационные параметры кластера.

Конфигурационные параметры кластера перечислены ниже:

- ***Reliability control*** – параметры этой секции предназначены для настройки тайм-аута ожидания разблокировки пакетного менеджера DNF. Отображается, если активирован переключатель ***Show advanced*** в верхней части окна. В Ubuntu после перезагрузки хоста запускается обновление, что приводит к блокировке DNF.

Имя	Описание	Значение по умолчанию
retries	Максимальное число повторных попыток	80
delay	Тайм-аут между повторными попытками (в секундах)	30
timeout	Тайм-аут проверки сетевых портов (в секундах)	7200

- Пути к репозиториям, которые будут использованы в процессе установки в зависимости от операционной системы кластера (CentOS 7, Red Hat 7, Ubuntu 22, ALT Linux 8, Astra Linux, RedOS 7). На серверах кластера регистрируются только отмеченные репозитории.

Репозитории

Название репозитория	Параметр	Описание	Значение по умолчанию
monitoring	desc	Описание репозитория	Monitoring components url
	url	Путь к репозиторию Arenadata	Указано в зависимости от ОС
	enabled	Включает использование репозитория	true
ADS	desc	Описание репозитория	ADS repository
	url	Путь к репозиторию Arenadata	Указано в зависимости от ОС
	enabled	Включает использование репозитория	true
prometheus_repo	desc	Описание репозитория	Prometheus repository
	url	Путь к репозиторию Arenadata	Указано в зависимости от ОС
	enabled	Включает использование репозитория	true
arenadata_java	desc	Описание репозитория	Arenadata repository for OpenJDK
	url	Путь к репозиторию Arenadata	Указано в зависимости от ОС
	enabled	Включает использование репозитория. Если установить значение `true` перед действием ***Install***, ***Add/Remove components*** или ***Update*** для кластера или сервиса, на хостах будет установлена последняя версия Java 17	false
ranger plugins	desc	Описание репозитория	Arenadata Ranger plugins repository
	url	Путь к репозиторию Arenadata	Указано в зависимости от ОС
	enabled	Включает использование репозитория	true

- ***Precheck packages*** – включает проверку доступности пакета перед установкой.
- ***Docker registry url*** – указывает размещение Arenadata Docker Registry. Отображается, если активирован переключатель ***Show advanced*** в верхней части окна.
- ***Znode path for cluster in Zookeeper service*** – путь Znode для кластера ADS в сервисе Zookeeper. Отображается, если активирован переключатель ***Show advanced*** в верхней части окна.
- ***BASIC Authentication*** – конфигурация для аутентификации BASIC.
  Описание включения аутентификации приведено в статье Basic-аутентификация.
- ***SASL_PLAINTEXT Authentication*** – конфигурация для аутентификации SASL PLAINTEXT.
  Описание включения аутентификации приведено в статье SASL_PLAINTEXT.
- ***Kerberos*** – конфигурация для аутентификации по протоколу Kerberos.
  Описание включения аутентификации приведено в статьях:
  - MIT Kerberos
  - Kerberos с MS Active Directory
  - Kerberos с Samba
- ***java.io.tmpdir*** – системное свойство Java, которое определяет, куда JVM записывает временные файлы.
- ***Custom JDK settings*** – переключатель, определяющий необходимость использования кастомного пути к Java для сервисов кластера ADS. При активации переключателя становится доступно для заполнения поле ***JAVA_HOME***. В нем требуется указать путь к работающей Java. В выбранном пути должна содержаться папка *bin* с бинарным файлом Java, например: */usr/lib/jvm/jre-openjdk/*. При пустом значении поля ***JAVA_HOME*** для поиска Java используется системный путь по умолчанию: */usr/lib/jvm/java-1.17.0-openjdk.amd64/* (для всех операционных систем, кроме Centos, RedHat и AstraLinux).
- ***Cluster UUID*** – отображает универсальный уникальный идентификатор кластера (universally unique identifier).

Конфигурирование кластера

Configuration groups

Вкладка ***Configuration → Configuration groups*** предназначена для настройки конфиг-групп кластера.

Ansible settings

Вкладка ***Configuration → Ansible settings*** предназначена для ввода конфигурационных настроек Ansible на уровне кластера. Вкладка доступна начиная с версии ADCM 2.2.0.

Параметры Ansible

Имя	Описание	По умолчанию
forks	Число параллельных процессов, генерируемых при коммуникации с удаленными хостами	5

Установка кластера

Д. Ольга Семин, Дарья Барышева

Содержание

Назад к содержанию

Шаг 1. Запуск установки
- Установка всего кластера
- Установка отдельных сервисов

Шаг 2. Просмотр результатов установки

Шаг 3. Проверка статуса кластера после установки

Шаг 1. Запуск установки

Существует два способа установки:

1. **Установка всего кластера**. В этом методе все сервисы кластера устанавливаются автоматически друг за другом после применения действия **Install** к кластеру.
2. **Установка отдельных сервисов**. В этом методе каждый сервис устанавливается вручную путем применения к нему одного или нескольких действий. Этот способ также можно использовать для добавления новых сервисов в уже развернутый кластер.

В обоим методам каждый сервис запускается автоматически после установки.

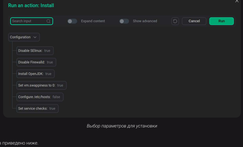
ВАЖНО

Установка должна производиться на чистые хосты без предустановленного подобного ПО.

Установка всего кластера

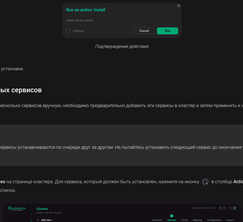
Чтобы установить все сервисы кластера за одну операцию, выполните шаги:

1. Примените действие **Install** к кластеру, выбрав/кнопку на странице **Clusters**, нажав на иконку ▶ в столбце **Actions**



Переход к установке всего кластера

2. В открывшемся диалоговом окне выберите необходимые параметры для установки и нажмите **Run**.

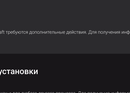


Выбор параметров для установки

Назначение параметров приведено ниже.

Параметр	Описание
Disable SELinux	Отключение SELinux на добавляемых хостах. Для того чтобы данная настройка применилась, после завершения операции **Install** необходимо перезагрузить хосты вручную
Disable Firewalld	Выключение firewalld на хостах
Install OpenJDK	Установка пакета java-1.8.0-openjdk на хостах
Set vm.swappiness to 0	Отключение аварийку на хостах
Configure /etc/hosts	Запись добавляемых нод в /etc/hosts на хостах кластера. Данную опцию рекомендуется отключить, если настроен DNS
Set service checks	Включит проверку сервиса при запуске действия

3. Подтвердите действие в открывшемся окне:



Подтверждение действия

4. Дождитесь завершения установки

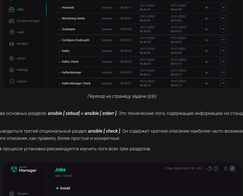
Установка отдельных сервисов

Чтобы установить один или несколько сервисов вручную, необходимо предварительно добавить эти сервисы в кластер и затем применить к ним действия, описанные ниже.

ВНИМАНИЕ

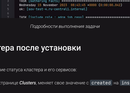
Обратите внимание, что сервисы устанавливаются по очереди друг за другом. Не пытайтесь установить следующий сервис до окончания установки предыдущего

1. Откройте вкладку **Services** на странице кластера. Для сервиса, который должен быть установлен, нажмите на иконку ▶ в столбце **Actions** и выберите действие **Install** из выпадающего списка.



Переход к установке отдельных сервисов

2. В открывшемся диалоговом окне выберите необходимые параметры для установки и нажмите **Run**.
3. Подтвердите действие в открывшемся окне:



Подтверждение действия

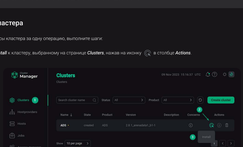
4. Дождитесь завершения установки и убедитесь, что статус сервиса изменился с `created` на значение `installed`.
5. Повторите предыдущие шаги для всех сервисов, которые необходимо установить.

ВАЖНО

- *При установке сервисов вручную важно соблюдать следующий порядок:*
  - ZooKeeper;
  - Kafka, NiFi (по одному);
  - другие сервисы.
- Во время установки сервиса Kafka в режиме KRaft требуются дополнительные действия. Для получения информации обратитесь к статье Установка Kafka в режиме KRaft.

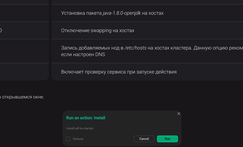
Шаг 2. Просмотр результатов установки

ADCM запускает отдельную задачу (job) для установки, как и для любого другого процесса. Для получения информации о статусе задач предназначена страница **Jobs**



Страница Jobs

Если установка завершается успешно, соответствующая ей задача на странице **Jobs** переходит из статуса `running` в `success`, в противном случае – в статус `failed`. Чтобы получить более детальную информацию о возможных ошибках, можно перейти на страницу задачи, кликнув по ней в списке задач на странице **Jobs**.

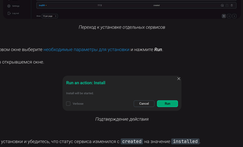


Переход на страницу задачи (job)

Страница задачи содержит два основных раздела: **ansible [stdout]** и **ansible [stderr]**. Это технические логи, содержащие информацию из стандартных I/O потоков ansible/ansible-playbook.

В дополнение к ним может выводиться третий специальный раздел **ansible [check]**. Он содержит краткое описание наиболее часто возникающих ошибок. По сравнению с предыдущими разделами, эти описания, как правило, более простые и конкретные.

При возникновении ошибок в процессе установки рекомендуется изучить логи всех трех разделов.

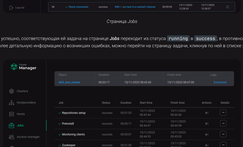


Подробности выполнения задачи

Шаг 3. Проверка статуса кластера после установки

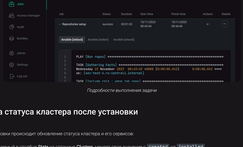
В результате успешной установки происходит обновление статуса кластера и его сервисов:

- Статус кластера, отображаемый в столбце **State** на странице **Clusters**, меняет свое значение с `created` на значение `installed`



Статус кластера после успешной установки

- Статус сервисов, отображаемый на вкладке **Services** страницы кластера, меняет свое значение с `created` на значение `installed`.



Статус сервисов кластера после успешной установки

Установка мониторинга

Ольга Семме

Назад к содержанию

Мониторинг — опциональный, но рекомендуемый компонент ADS, который можно настроить, используя один из следующих подходов:

- добавить в кластер ADS сервис Monitoring (доступен начиная с версии 3.6.2.2.b1);
- создать кластер мониторинга и интегрировать его с кластером ADS (поддержка этого способа установки мониторинга будет прекращена в будущих релизах ADS).

Сервис мониторинга

Артём Семенов

Содержание

Обзор
Шаг 1. Добавление сервиса
Шаг 2. Добавление компонентов
Шаг 3. Настройка сервиса
Шаг 4. Установка сервиса
Шаг 5. Проверка результатов
Данные в формате Prometheus
Веб-интерфейс Prometheus
Веб-интерфейс Grafana

Назад к содержанию

Обзор

При добавлении сервиса Monitoring в кластер K0S вся часть установляется на control plane. Этот компонент устанавливает с control plane системные метрики, которые будет собирать Prometheus. Prometheus, как и Grafana, также установлен на control plane. Prometheus собирает метрики с control plane, worker nodes, API, kubelet, Konnectivity, Calico, CoreDNS. Для настройки Grafana и формата Prometheus см. пункты с похожими темами (информер, развёртывание и конфигурирование Prometheus сервера).

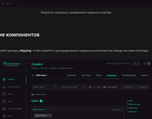
Для работы Prometheus, Grafana, которые собирают данные с кластера, нужно в веб-интерфейсе Prometheus или Grafana:

ПРИМЕЧАНИЕ

- Если узел, где установлен Prometheus, недоступен, метрики с кластера не будут собираться. Для этого нужно настроить [Prometheus federation](https://prometheus.io/docs/prometheus/latest/federation/) — механизм, позволяющий одному Prometheus-серверу собирать данные с других Prometheus-серверов. Это позволяет создавать иерархии и объединять данные с нескольких кластеров K0S.
- Вы также можете использовать [Thanos](https://thanos.io/), чтобы перенести все метрики из сервиса Prometheus, развёрнутого в K0S, в свой Prometheus.

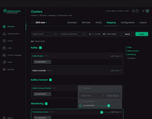
Шаг 1. Добавление сервиса

1. В интерфейсе K0S control plane в сервисе **Clusters** в элементе по имени кластера K0S. Затем во вкладке **Services** нажмите кнопку **Add service**.



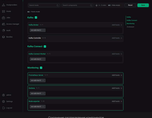
Переход к добавлению сервиса

2. В открывшемся окне выберите сервис **Monitoring** и нажмите **Add**.



Выбор сервиса

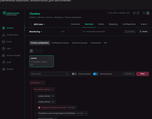
В результате добавленный сервис отобразится во вкладке **Services**.



Результат добавления сервиса в кластер

Шаг 2. Добавление компонентов

1. На странице настроек сервиса во вкладке **Mapping** чтобы перейти к расширенному справочнику компонентов между настройками сервиса.

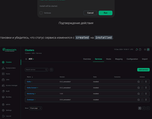


Переход к расширенному справочнику компонентов

Компоненты сервиса Monitoring

Компонент	Описание
Prometheus Server	Развёртывает в K0S сервер Prometheus, который выполняет следующие функции: • Агрегирование для всех кластеров, собирающих метрики в кластер. • хранилище всех метрик кластера. • Интерфейс для запросов на основе собранных метрик.
Grafana	Позволяет визуализировать метрики K0S в виде графиков и диаграмм, организованных в дашборды.
Node exporter	Экспортирует в Prometheus метрики узлов K0S.

2. Нажмите кнопку напротив компонента сервиса Monitoring — элемента **Add hosts** и выберите нужный хост из открывшегося окна.



Выбор хоста для компонента

ПРИМЕЧАНИЕ

Компоненты Prometheus Server не рекомендуется устанавливать на хост с K0S — это может работать для теста, но не для продакшена. Prometheus Server потребляет много ресурсов CPU и RAM, что может привести к нестабильной работе K0S. Рекомендуется выделять для Prometheus Server отдельные worker-машины.

3. После завершения распределения компонентов нажмите **Save**, чтобы сохранить изменения.

Сохранение распределения компонентов

Шаг 3. Настройка сервиса

1. Откройте вкладку **Services** на странице кластера и нажмите на имя сервиса **Monitoring** в столбце **Name**.

Переход к настройке сервиса

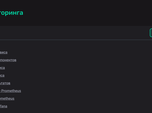
2. В открывшемся окне настроек сервиса Monitoring — конечном параметре конфигурации сервиса, в разделе **Monitoring** опции [конфигурационного справочника K0S](). Поля, перечисленные ниллсами, обязательны для заполнения.

Настройка сервиса Monitoring

После указания всех необходимых параметров нажмите **Save**.

Шаг 4. Установка сервиса

1. На вкладке **Services** в списке сервисов Monitoring появился по иконке **Actions** в контекстном меню выберите пункт **Install**.



Переход к установке сервиса

2. Подтвердите действие в открывшемся окне.

Подтверждение действия

3. Дождитесь завершения установки и убедитесь, что статус сервиса изменился на **installed** — **Uninstalled**.

Процесс установки завершён

Чтобы проконтролировать процесс выполнения установки сервиса на проблемы/ошибки, ошибки в случае их возникновения, выберите пункт **Jobs** в левом навигационном меню и нажмите на ссылку **Install** в поле задач **Jobs**.

Страница задачи Install

Шаг 5. Проверка результатов

Сервис Monitoring предоставляет метрики кластера и его элементов. По завершении установки необходимо убедиться в следующем:

- Метрики собираются по всем хостам кластера. Для этого в веб-интерфейсе Prometheus перейдите в раздел **Status** — **Targets** и убедитесь, что все цели имеют статус **UP**.
- С кластера корректно поступают системные метрики, а также метрики сервиса K0S — см. [Запрос системных метрик кластера K0S]().

Чтобы проверить, что данные с кластера поступают в веб-интерфейс Prometheus, в меню интерфейса веб-интерфейса Prometheus и Grafana.

ПРИМЕЧАНИЕ

Для работы мониторинга K0S на странице [Configuration](#) сервиса Kuffy перейдите по разделу **API Monitoring** и активируйте компоненты. После этого перезагрузите сервис.

Данные в формате Prometheus

В формате Prometheus можно получить метрики K0S, а также сервисов Kuberntes, развёрнутых в кластере. Для доступа к метрикам, которые предоставляет Prometheus, используется **Prometheus settings** — предоставляемый для настройки сервиса Prometheus **Kuyr-exporter settings**, предназначенный для настройки сервиса Prometheus.

Prometheus-сервер для Kuberntes-кластера K0S работает на [10.10.10.247](). Чтобы получить данные Prometheus:

1. Перейдите в веб-интерфейс Prometheus и нажмите на ссылку **Prometheus** в разделе **Prometheus**.

Переход к разделу Prometheus

2. В открывшемся окне выберите раздел **Prometheus settings** и нажмите на ссылку **Prometheus**.

Переход к разделу Prometheus

3. В поле **Expression** можно ввести название метрики, например, **Cluster** — в интерфейсе будут показаны значения этой метрики на всех кластерах K0S.

Веб-интерфейс Prometheus

Веб-интерфейс Prometheus

В веб-интерфейсе Prometheus можно получить метрики K0S, а также сервисов Kuberntes, развёрнутых в кластере. Для доступа к метрикам, которые предоставляет Prometheus, используется **Prometheus settings** — предоставляемый для настройки сервиса Prometheus **Kuyr-exporter settings**, предназначенный для настройки сервиса Prometheus.

Prometheus-сервер для Kuberntes-кластера K0S работает на [10.10.10.247](). Чтобы получить данные Prometheus:

1. Перейдите в веб-интерфейс Prometheus и нажмите на ссылку **Prometheus** в разделе **Prometheus**.

Переход к разделу Prometheus

2. В открывшемся окне выберите раздел **Prometheus settings** и нажмите на ссылку **Prometheus**.

Переход к разделу Prometheus

3. В поле **Expression** можно ввести название метрики, например, **Cluster** — в интерфейсе будут показаны значения этой метрики на всех кластерах K0S.

Веб-интерфейс Prometheus

Веб-интерфейс Grafana

В веб-интерфейсе Grafana можно получить метрики K0S, а также сервисов Kuberntes, развёрнутых в кластере. Для доступа к метрикам, которые предоставляет Grafana, используется **Grafana settings** — предоставляемый для настройки сервиса Grafana **Kuyr-exporter settings**, предназначенный для настройки сервиса Grafana. Grafana-сервер для Kuberntes-кластера K0S работает на [10.10.10.247:3000](). Для доступа к метрикам, которые предоставляет Grafana, используется **Grafana settings** — предоставляемый для настройки сервиса Grafana **Kuyr-exporter settings**, предназначенный для настройки сервиса Grafana.

1. В открывшемся окне выберите раздел меню **Home** — **Dashboards** и разверните раздел **Dashboard «K0S_cluster_name»**. В открывшемся окне выберите один из доступных дашбордов для просмотра метрик кластера или сервисов или нажмите на ссылку **New** для создания нового дашборда.

Переход к интерфейсу кластера K0S в Grafana

Установка мониторинга

Дарья Барышева

Назад к содержанию

Мониторинг является опциональным компонентом (однако, рекомендуемым). ADCM поддерживает настройку мониторинга на базе трех сервисов: Diamond, Graphite и Grafana.

Для процесса **online**-установки необходим отдельный кластер мониторинга. Шаги для установки мониторинга в этом случае приведены ниже. Их можно выполнить до и после установки кластера ADS, кроме заключительного шага Интеграция с кластером ADS, который может быть выполнен только на базе добавленного кластера ADS.

- Создание кластера
- Добавление сервисов
- Добавление хостов в кластер
- Добавление компонентов
- Настройка сервисов
- Установка кластера
- Интеграция с кластером ADS

ВНИМАНИЕ

Начиная с версии 3.6.2.2.b1 не рекомендуется настраивать мониторинг ADS через отдельный кластер мониторинга, так как с этого момента поддержка данной функциональности приостановлена и будет полностью прекращена в будущих релизах ADS. Рекомендуемый способ мониторинга кластера ADS — использовать сервис Monitoring.

Создание кластера

Дарья Барышева

Содержание

Назад к содержанию

Шаг 1. Загрузка бандла

Шаг 2. Загрузка бандла в ADCM

Шаг 3. Создание кластера на базе загруженного бандла

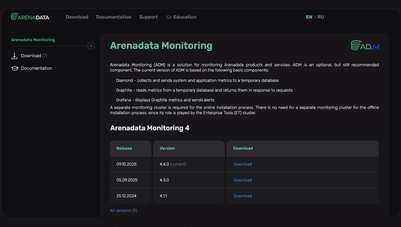
В ADCM **кластер** означает набор сервисов, запущенных на группе хостов.

Дистрибутивы кластеров для ADCM поставляются в виде бандлов. В общем случае бандл представляет обычный архив, описывающий логику развертывания кластера.

Шаги для добавления нового кластера приведены ниже.

Шаг 1. Загрузка бандла

1. Зайдите на сайт https://network.arenadata.io/ и выберите раздел ***Arenadata Monitoring***.
2. На следующей странице загрузите необходимый бандл.

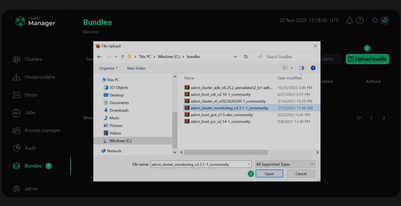


Страница загрузки бандлов Arenadata Monitoring

Шаг 2. Загрузка бандла в ADCM

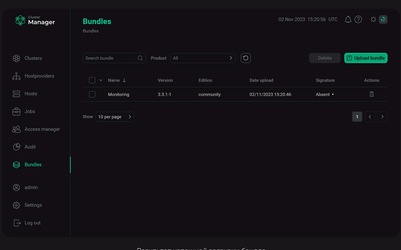
Чтобы загрузить бандл в ADCM:

1. Выберите пункт ***Bundles*** в левом навигационном меню и нажмите ***Upload bundle***.
2. Выберите бандл в открывшемся диалоговом окне.



Переход к загрузке бандла

3. После успешной загрузки бандл отображается на странице ***Bundles***.



Результат успешной загрузки бандла

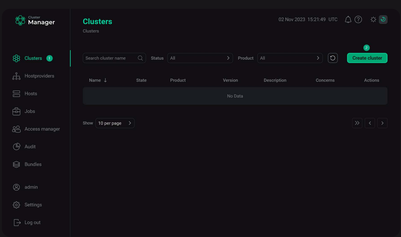
Шаг 3. Создание кластера на базе загруженного бандла

ВАЖНО

На этапе создания кластера через web-интерфейс ADCM генерируется только сам кластер. Это означает добавление записи о кластере в ADCM — сервисы на этом шаге еще не устанавливаются.

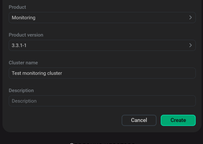
Для создания кластера необходимо выполнить шаги:

1. Выберите пункт ***Clusters*** в левом навигационном меню и нажмите ***Create cluster***.



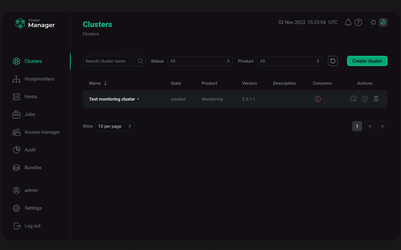
Переход к созданию кластера

2. В открывшемся окне требуется описать новый кластер:
   - Выберите загруженный бандл в поле ***Product***.
   - Выберите версию бандла в поле ***Product version***. Несколько версий становятся доступны после загрузки нескольких бандлов для одного типа продуктов.
   - Введите имя кластера в поле ***Cluster name***.
   - При необходимости добавьте описание кластера в поле ***Description***.
   - Установите флаг ***I accept Terms of Agreement*** для подписания лицензионного соглашения. Флаг отображается только для бандлов версии Enterprise. Чтобы ознакомиться с текстом соглашения, перейдите по ссылке ***Terms of Agreement***. Подписать соглашение также возможно на странице ***Bundles***, нажав на соответствующий продуктовый бандл.
   - Нажмите ***Create***.



Создание кластера

3. После успешного добавления кластер отображается на странице ***Clusters***.



Результат успешного добавления кластера

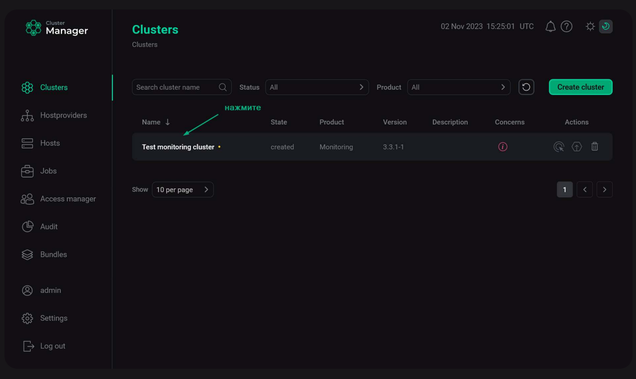
Добавление сервисов

Дарья Барышева

Назад к содержанию

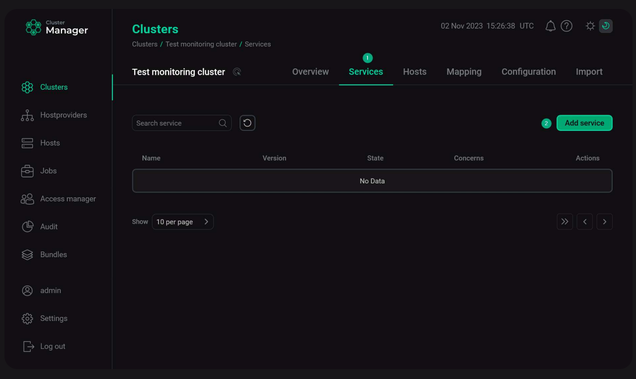
В ADCM **сервис** означает программное обеспечение, выполняющее некоторую функцию. Для добавления сервисов в кластер:

1. Выберите кластер на странице ***Clusters***. Для этого нажмите на имя кластера в столбце ***Name***.



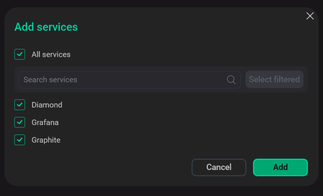
Выбор кластера

2. Откройте вкладку ***Services*** на странице кластера и нажмите ***Add service***.



Переход к добавлению сервисов

3. В открывшемся окне выберите сервисы для добавления в кластер и нажмите ***Add***.



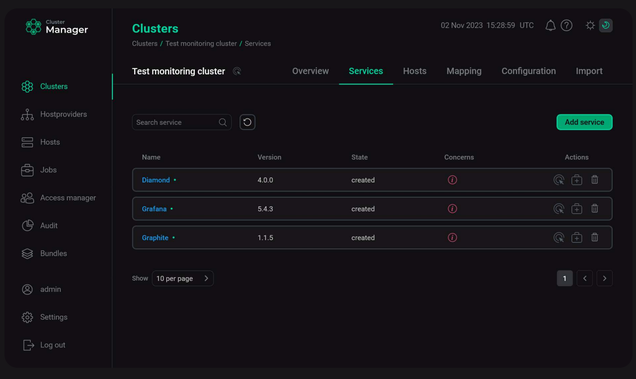
Выбор сервисов

Краткое описание доступных сервисов приведено ниже.

Сервисы, которые необходимо добавить в кластер мониторинга

Сервис	Назначение
Diamond	Python-сервис, используемый для сбора системных метрик с хостов, подключенных к системе мониторинга. Собранные метрики обрабатываются в Carbon (backend-сервисах Graphite)
Graphite	Состоит из нескольких компонентов: ■ **Carbon** — набор сервисов для обработки метрик, поступающих из Diamond. ■ **Whisper DB** — временная файловая база данных, где хранятся собранные метрики. ■ **Web service** — web-сервис, позволяющий получать и визуализировать метрики из базы данных Whisper DB.
Grafana	Web-сервис, предназначенный для создания графиков и оповещений для визуализации метрик, полученных из Graphite. По умолчанию использует порт `3000`

4. В результате успешно добавленные сервисы отображаются на вкладке ***Services***.



Результат успешного добавления сервисов в кластер

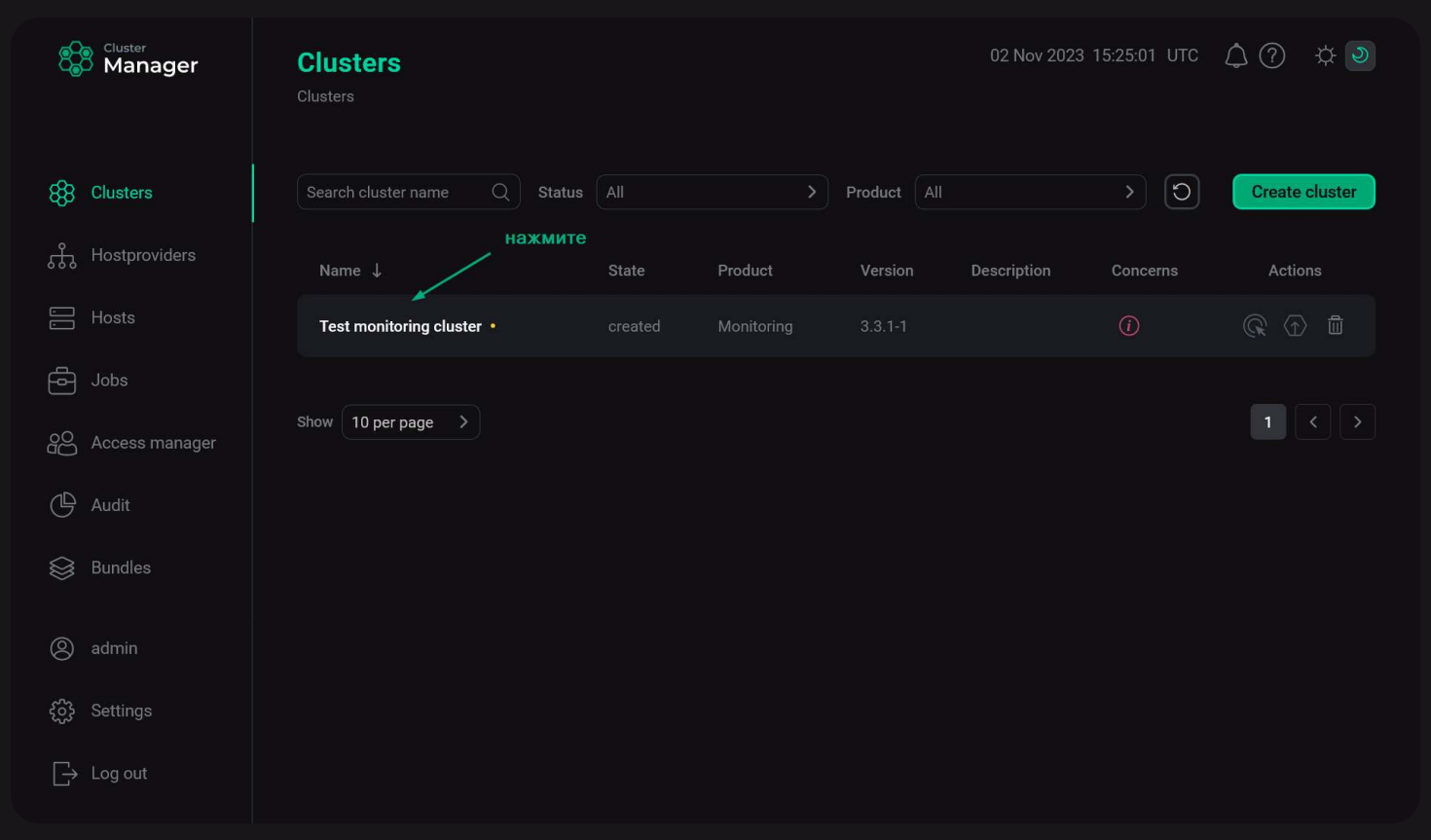
Добавление хостов в кластер

Дарья Барышева

Назад к содержанию

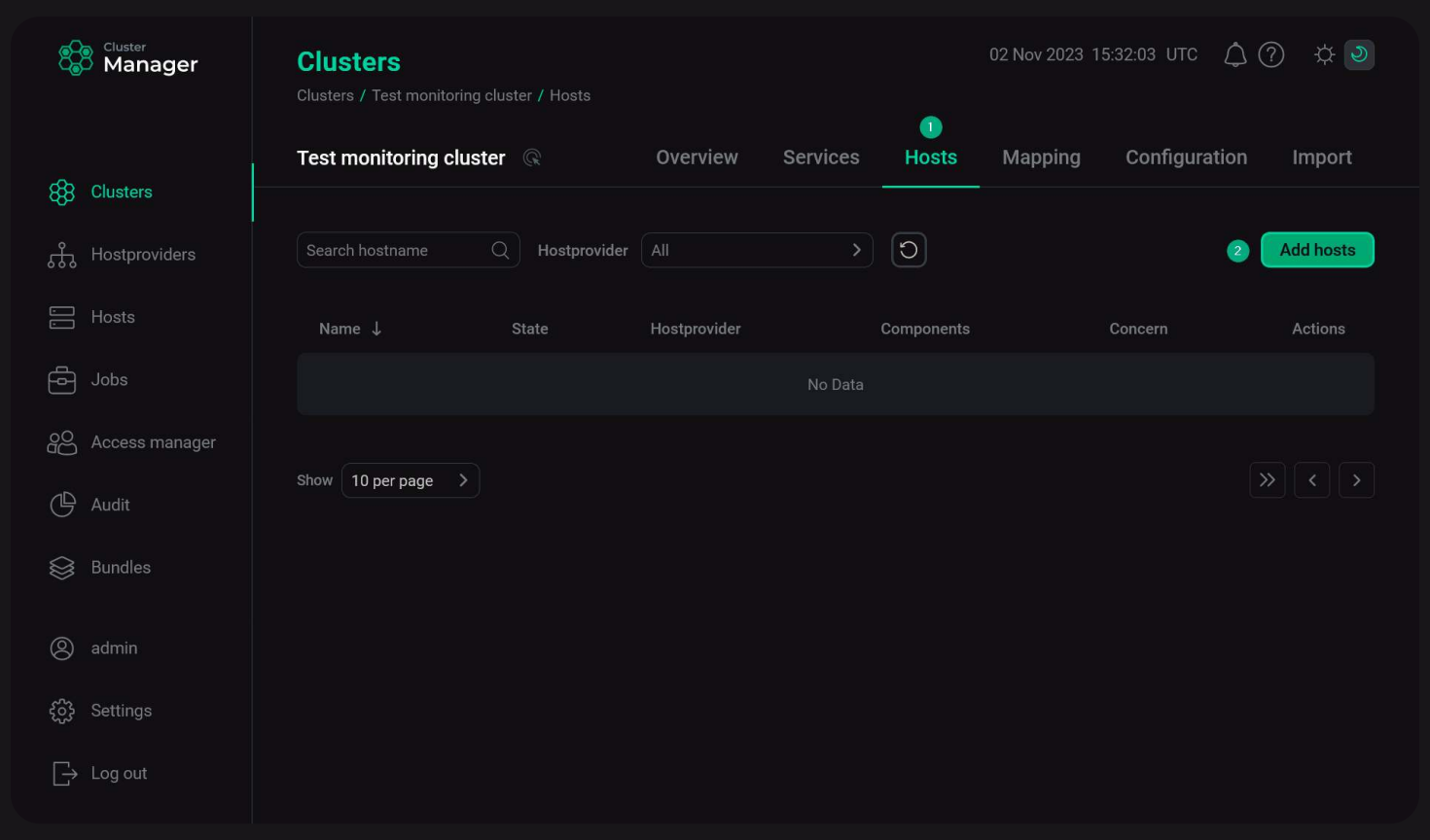
Для добавления хостов в кластер:

1. Выберите кластер на странице ***Clusters***. Для этого нажмите на имя кластера в столбце ***Name***.



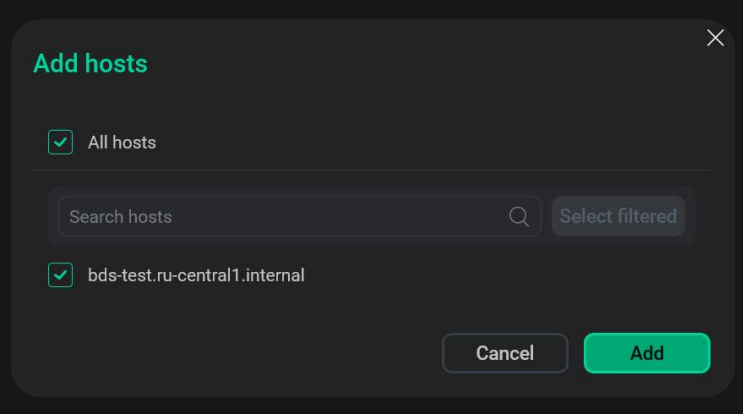
Выбор кластера

2. Откройте вкладку ***Hosts*** на странице кластера и нажмите ***Add hosts***.



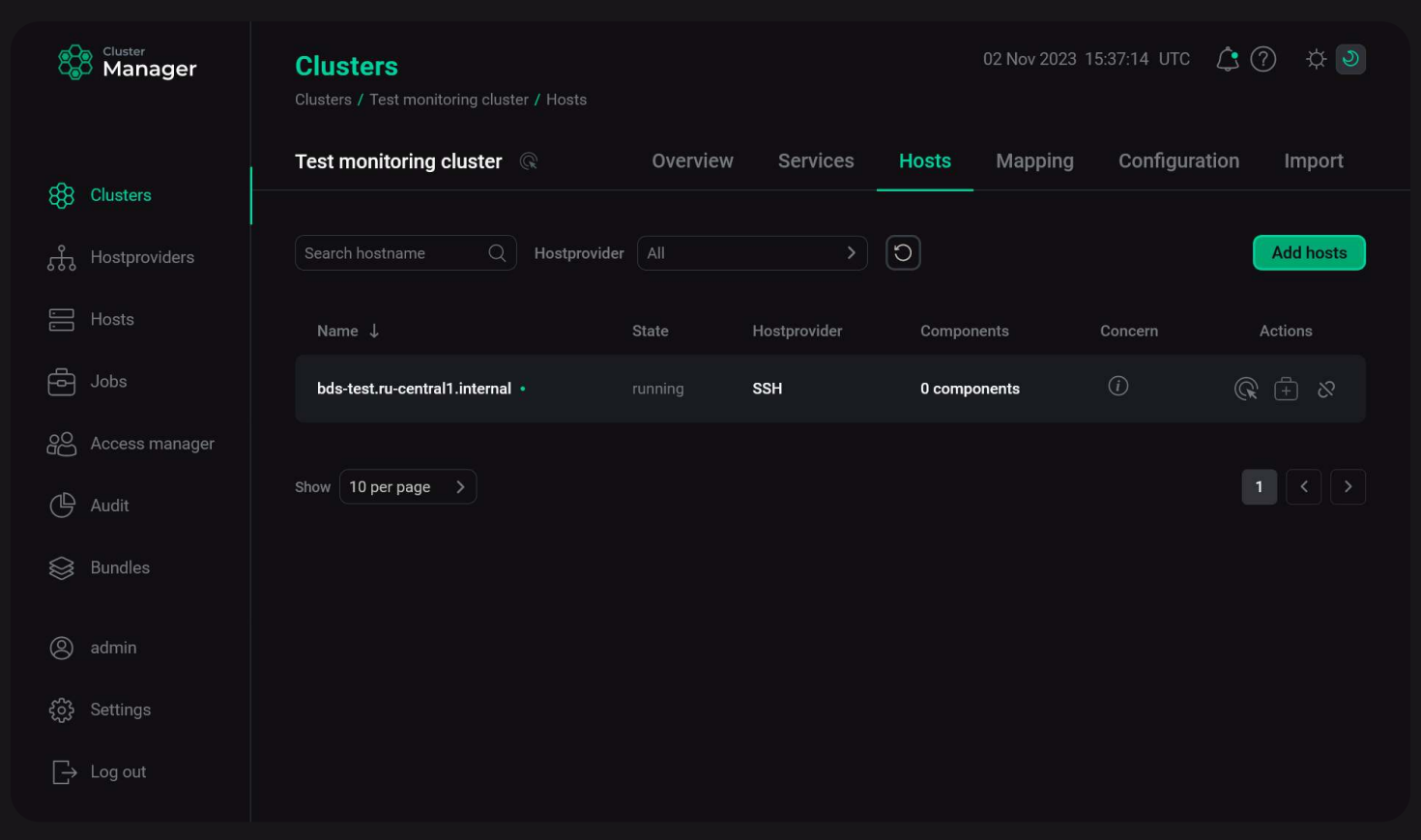
Переход к добавлению хостов

3. В открывшемся окне выберите хосты для добавления в кластер и нажмите ***Add***.



Выбор хостов

4. В результате успешно добавленные хосты отображаются на вкладке ***Hosts***.



Результат успешного добавления хостов в кластер

Добавление компонентов

Дарья Барышева

Назад к содержанию

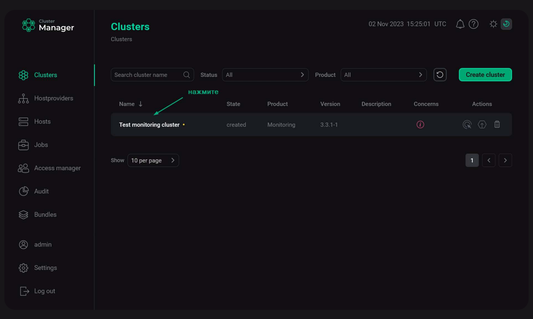
В ADCM **компонент** означает часть сервиса, которая должна быть развернута на одном или нескольких хостах кластера.

ВАЖНО

Изначально на хостах кластера нет ни одного компонента — распределение компонентов между хостами должно быть проведено вручную.

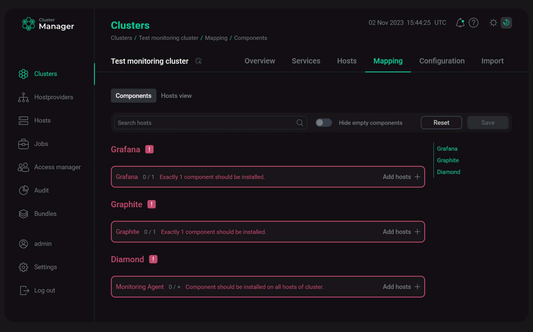
Для добавления компонентов на хосты кластера:

1. Выберите кластер на странице *Clusters*. Для этого нажмите на имя кластера в столбце *Name*.



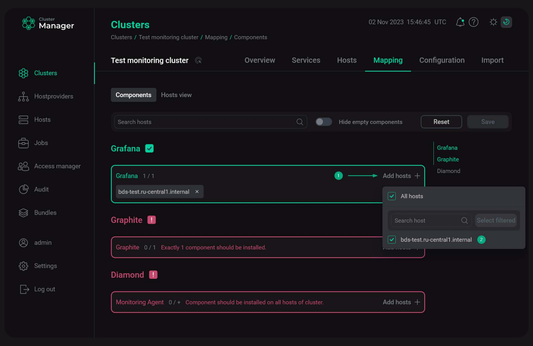
Выбор кластера

2. Откройте вкладку *Mapping* на странице кластера.



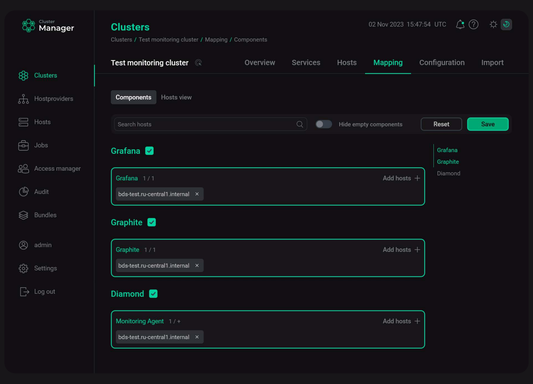
Переход к распределению сервисных компонентов

3. Распределите все компоненты между хостами кластера. Для назначения хостов компоненту нажмите *Add hosts* и выберите хосты в открывшемся окне.



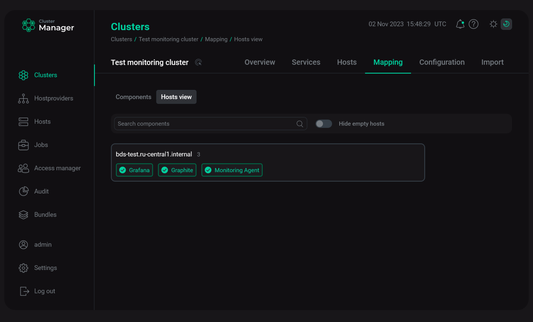
Выбор хостов для компонента

4. После завершения распределения компонентов нажмите *Save*.



Сохранение распределения компонентов

5. Для перехода к альтернативному варианту просмотра — с разбиением по хостам — можно открыть вкладку *Hosts view*.



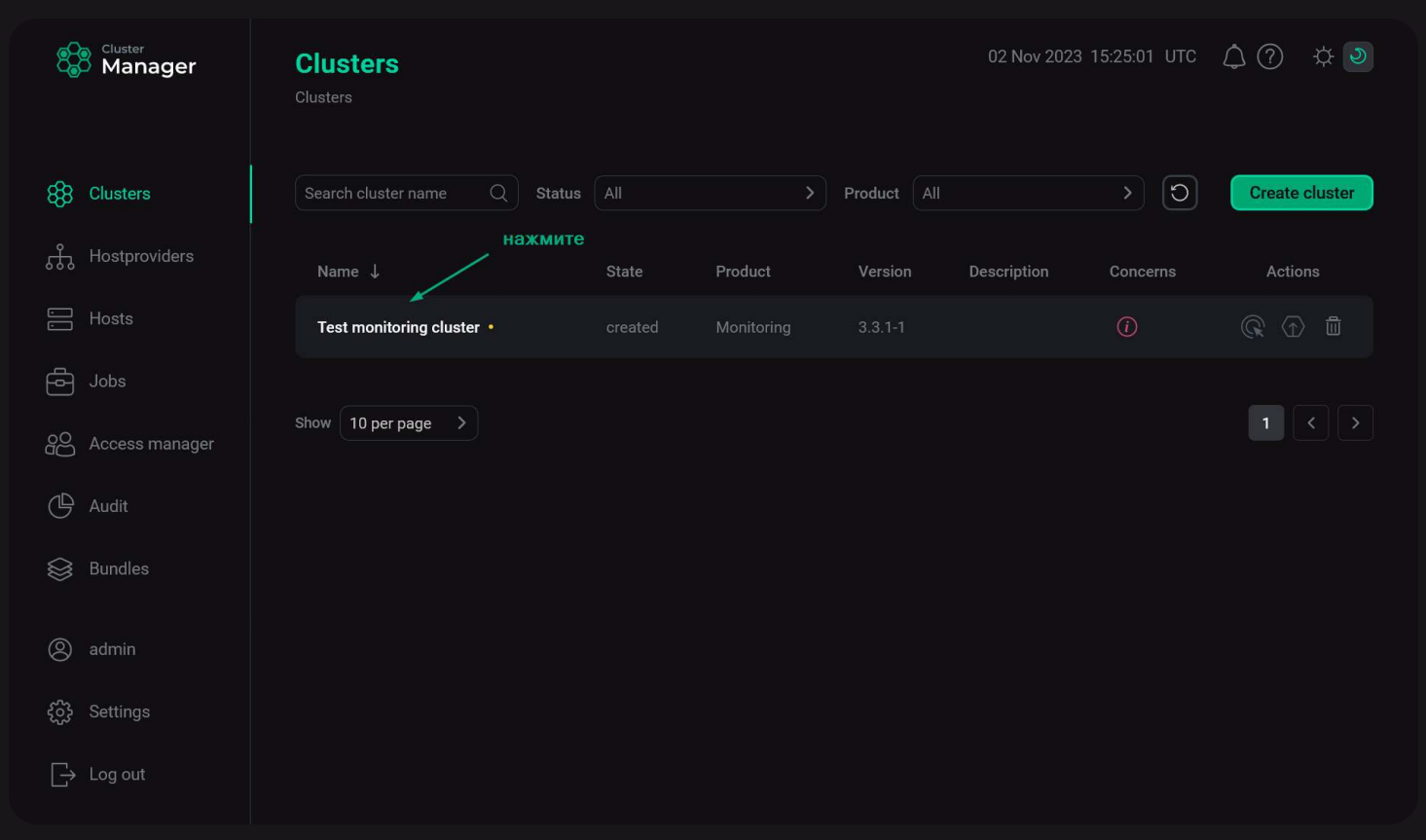
Просмотр распределения по хостам

Настройка сервисов

Дарья Барышева

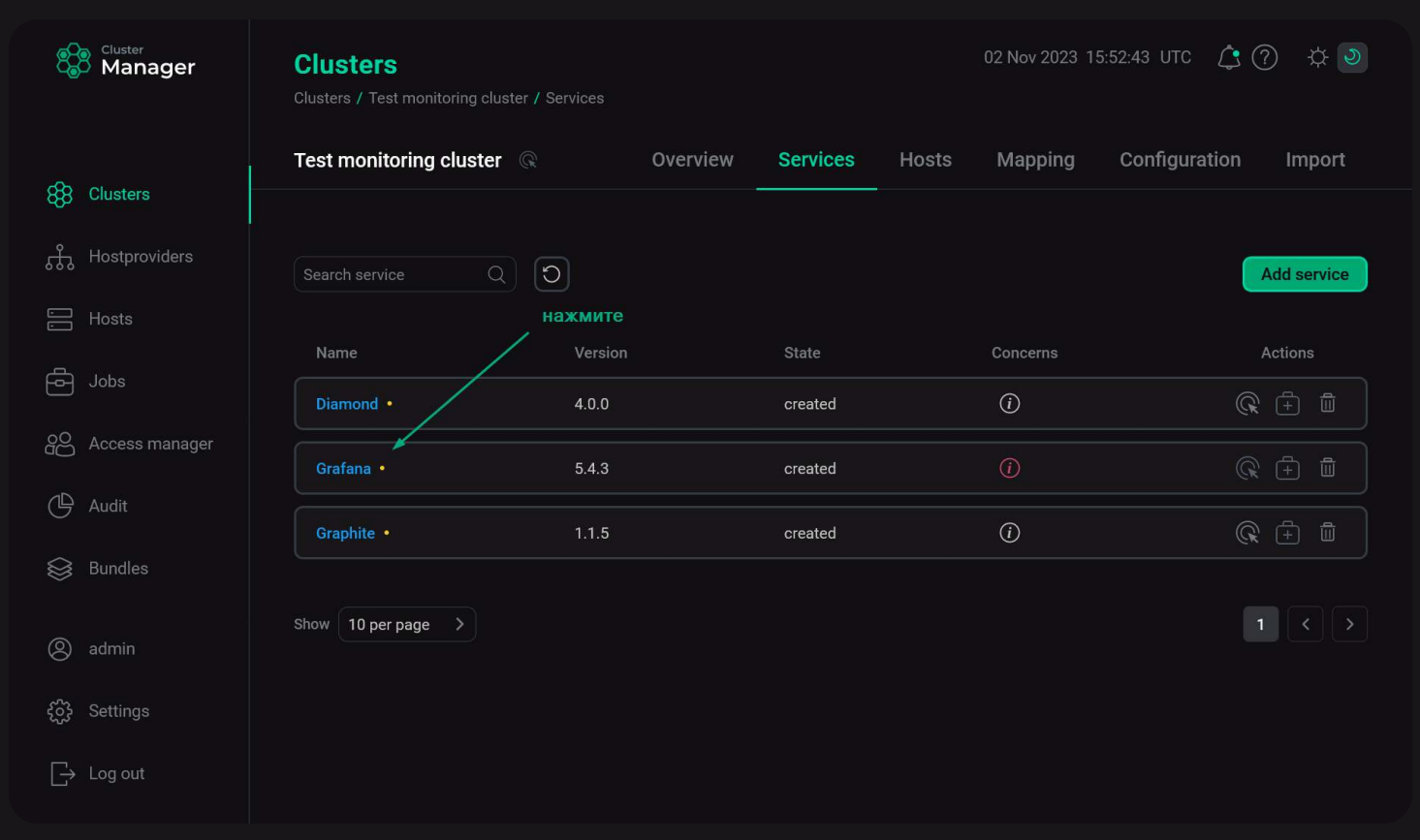
Назад к содержанию

1. Выберите кластер на странице ***Clusters***. Для этого нажмите на имя кластера в столбце ***Name***.



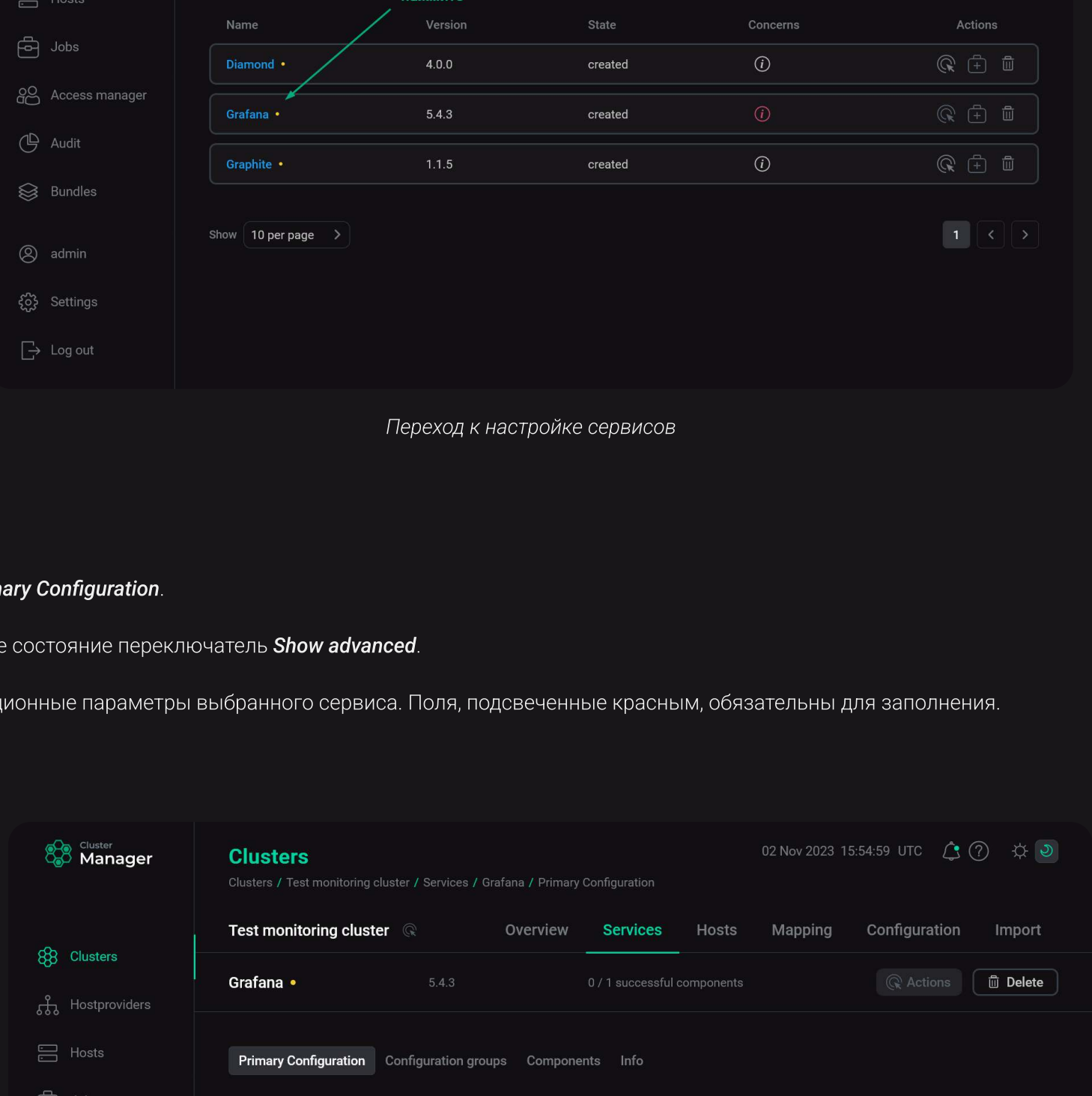
Выбор кластера

2. Откройте вкладку ***Services*** на странице кластера. Нажмите на имя сервиса в столбце ***Name***.



Переход к настройке сервисов

3. В открывшемся окне:
   - Откройте вкладку ***Primary Configuration***.
   - Переведите в активное состояние переключатель ***Show advanced***.
   - Заполните конфигурационные параметры выбранного сервиса. Поля, подсвеченные красным, обязательны для заполнения.
   - Нажмите ***Save***.



Настройка сервиса

Установка кластера

Дарья Барышева

Содержание

Назад к содержанию

Шаг 1. Запуск установки

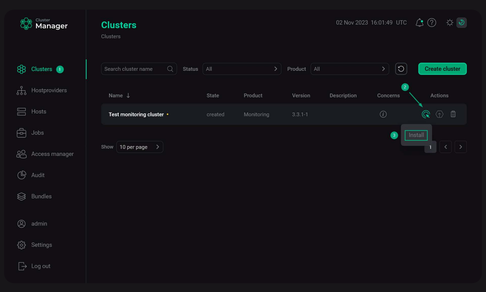
Шаг 2. Проверка результатов

- Graphite
- Grafana

Шаг 1. Запуск установки

Для установки кластера мониторинга необходимо выполнить шаги:

1. Запустите действие ***Install*** для кластера, выбранного на странице ***Clusters***, нажав на иконку в столбце ***Actions***. Подтвердите операцию в открывшемся окне. Все сервисы устанавливаются по очереди.



Переход к установке кластера мониторинга

2. Дождитесь завершения процесса установки.

ПРИМЕЧАНИЕ

В случае ошибок логи можно найти на странице ***Jobs***.

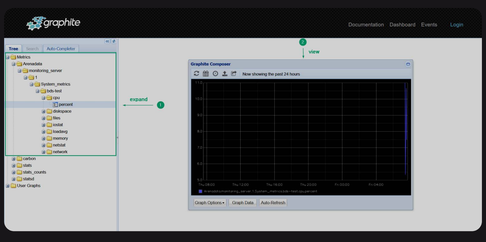
Шаг 2. Проверка результатов

Graphite

Чтобы проверить доступность web-интерфейса Graphite, необходимо выполнить шаги:

1. В адресной строке браузера введите адрес хоста, на котором развернут кластер мониторинга. В качестве номера порта укажите значение параметра ***Web-interface TCP port*** из раздела ***ip_and_ports***, определенное на этапе конфигурирования сервиса Graphite при установке кластера мониторинга. Значение `80` (используемое по умолчанию) можно не указывать.

2. Разверните дерево в левой части открывшейся формы. Здесь отображаются метрики System метрики с группировкой по хостам. Можно выбрать любую из метрик для просмотра динамики ее изменения в виде графика в правой части экрана.



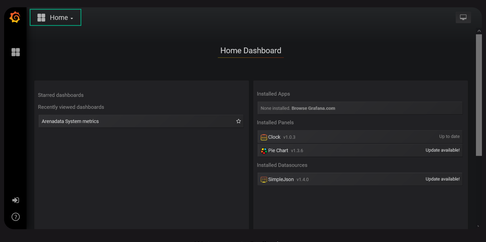
Web-интерфейс Graphite

Grafana

Чтобы проверить доступность web-интерфейса Grafana, необходимо выполнить шаги:

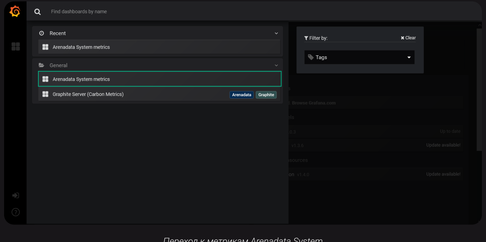
1. В адресной строке браузера введите адрес хоста, на котором развернут кластер мониторинга. В качестве номера порта укажите значение параметра ***Port*** из раздела ***ip_and_ports***, определенное на этапе конфигурирования сервиса Grafana при установке кластера мониторинга (по умолчанию `3000`).

2. Выберите пункт меню ***Home*** в открывшемся окне.



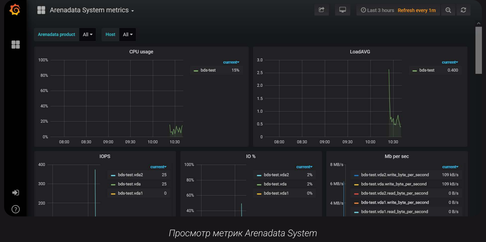
Web-интерфейс Grafana

3. Откройте любой из доступных дашбордов (dashboards). Например, ***Arenadata System metrics***.



Переход к метрикам Arenadata System

4. Проанализируйте доступные графики.



Просмотр метрик Arenadata System

Интеграция с кластером ADS

Константин Шарапов

Назад к содержанию

Содержание

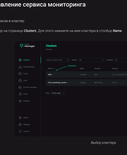
Шаг 1. Добавление сервиса мониторинга
Шаг 2. Распределение компонентов мониторинга
Шаг 3. Импорт настроек мониторинга
Шаг 4. Установка сервиса мониторинга
Шаг 5. Проверка результатов
- Graphite
- Grafana

Для настройки мониторинга нового кластера необходимо добавить отдельный сервис Monitoring Clients и интегрировать его с предыдущими созданным кластером мониторинга.
Шаги для интеграции кластера мониторинга и кластера ADS описаны ниже. Все действия выполняются на базе кластера ADS.

Шаг 1. Добавление сервиса мониторинга

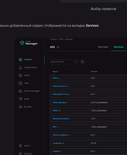
Для добавления сервиса в кластер:

1. Выберите кластер на странице **Clusters**. Для этого нажмите на имя кластера в столбце **Name**.



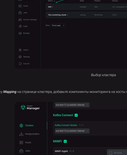
Выбор кластера

2. Откройте вкладку **Services** на странице кластера и нажмите **Add service**



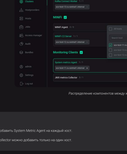
Переход к добавлению сервисов

3. Выберите сервис Monitoring Clients в открывшемся окне и нажмите **Add**



Выбор сервиса

4. В результате успешно добавленный сервис отобразится на вкладке **Services**.

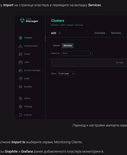


Результат успешного добавления сервиса в кластер

Шаг 2. Распределение компонентов мониторинга

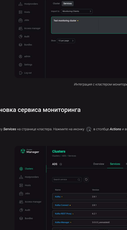
Для добавления компонентов на хосты кластера:

1. Выберите кластер на странице **Clusters**. Для этого нажмите на имя кластера в столбце **Name**.

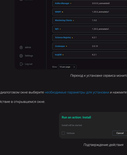


Выбор кластера

2. Откройте вкладку **Mapping** на странице кластера, добавьте компоненты мониторинга в их хосты кластера и нажмите **Save**



Распределение компонентов между хостами

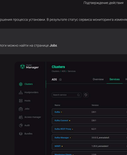


ВАЖНО

- Необходимо добавить System Metric Agent на каждый хост
- Для метрик Collector можно добавить только на один хост

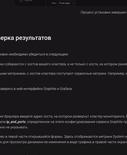
Шаг 3. Импорт настроек мониторинга

1. Выберите кластер на странице **Clusters**. Для этого нажмите на имя кластера в столбце **Name**.



Выбор кластера

2. Откройте вкладку **Import** на странице кластера и перейдите на вкладку **Services**.



Переход к настройкам импорта сервисов

3. В выпадающем списке **Import to** выберите сервис Monitoring Clients.
4. Выберите сервисы **Graphite** и **Grafana** (если добавляемого кластера мониторинга).
5. Нажмите **Import**

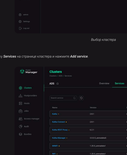
Интеграция с кластером мониторинга

Шаг 4. Установка сервиса мониторинга

1. Откройте вкладку **Services** на странице кластера. Нажмите на иконку ⚙ в столбце **Actions** и выберите действие **Install** в строке содержащей сервис Monitoring Clients.

Переход к установке сервиса мониторинга

2. В открывшемся диалоговом окне выберите [необходимые параметры для установки](#) и нажмите **Run**
3. Подтвердите действие в открывшемся окне.



Подтверждение действия

4. Дождитесь завершения процесса установки. В результате статус сервиса мониторинга изменит свое значение `STARTED` на `INSTALLED`

ПРИМЕЧАНИЕ

В случае ошибки, есть немного сайтов на странице **Jobs**.

Процесс установки завершен

Шаг 5. Проверка результатов

По завершении установки необходимо убедиться в следующем:

- Системные метрики собираются с хостов вашего кластера, а не только с хоста, на котором ранее был развернут кластер мониторинга.
- Хосты с системными метриками с хостов кластера поступают сервисные метрики. Например, хосты ADS обеспечивают метриками для своих сервисов как Kafka, NiFi, Ranger.

Обе группы метрик проверить в веб-интерфейсе Graphite и Grafana.

Graphite

1. В адресной строке браузера введите адрес хоста, на котором развернут хост мониторинга. В качестве номера порта укажите значение параметра **dist-interface.port** для ресурса **gl_web_ports**, предоставленное вам этим конфигурационным файлом (Graphite будет установлен на порт по умолчанию `8080`).
2. Разверните дерево в левой части открывшейся формы. Здесь отображаются метрики System и сервисные метрики с группировкой по хостам. Можно выбрать любую из метрик для просмотра значения во временном в виде графика в правой части экрана.

Новые хосты и сервисные метрики доступны в Graphite

Grafana

1. В адресной строке браузера введите адрес хоста, на котором развернут хост мониторинга. В качестве номера порта укажите значение параметра **Port** из ресурса **gl_web_ports**, предоставленное вам этим конфигурационным файлом (Grafana будет установлена на порт по умолчанию `3000`).
2. Выберите пункт меню **Home** в открывшемся окне.
3. Обратите внимание, что новые дашборды доступны. Можно выбрать один из них для просмотра сервисных метрик, поступающих на хосты кластера.

Переход к сервисным метрикам

4. Просматривайте доступные графики.

Просмотр сервисных метрик

Offline-установка

Назад к содержанию

Этот раздел посвящен **offline**-установке ADS. Данный способ установки подходит для хостов без доступа к Интернету.

Для offline-установки необходимо установить и настроить Arenadata Enterprise Tools (ET). ET — это решение, предоставляющее инфраструктуру для развертывания продуктов Аренадата в среде с ограниченным доступом в интернет. Решение является частью платформы данных Аренадата и предоставляет репозитории, необходимые для установки и обновления компонентов платформы.

Процесс offline-установки через ADCM включает следующие шаги.

ПРИМЕЧАНИЕ

Эта функция доступна только в Enterprise-версии ADS.

Шаг 1.

Установка ADCM

Шаг 2.

Подготовка хостов

Шаг 3.

Установка кластера ET

Шаг 4.

Установка кластера ADS

Шаг 5.

Установка мониторинга

Установка АЛСМ

Содержание

Шаг 1. Предварительные требования
- Требования к ядру и libc
- Установка Docker
- Установка ИК.Core
- Установка ИК.Node
- Установка и настройка nginx (ОС Astra Linux 1.8)

Шаг 2. Установка АЛСМ

Шаг 3. Запуск АЛСМ

Шаг 4. Проверка установки
- Проверка статуса Docker-контейнеров
- Проверка доступности по http и https
- Проверка сайта АЛСМ в браузере
- Проверка веб-интерфейса АЛСМ

Шаг 5. Обновление АЛСМ

Если вам нужна помощь с установкой — пишите в support@alarm.ru.

Шаг 1. Предварительные требования

Требования к ядру и libc

Для установки АЛСМ подходят дистрибутивы Linux, которые соответствуют следующим требованиям:

- Архитектура x86-64 (amd64)
- Ядро Linux:
  - версии 5
  - версии 6
  - не ниже 5.4
- glibc не ниже 2.31
- Поддержка cgroup v2

Проверить версию ядра и glibc можно командами:

```
uname -r
ldd --version | head -1
```

Установка Docker

Установите Docker Engine и Docker Compose Plugin по официальной инструкции для вашего дистрибутива:

- [Astra Linux](https://docs.docker.com/engine/install/debian/)
- [Debian](https://docs.docker.com/engine/install/debian/)
- [Ubuntu](https://docs.docker.com/engine/install/ubuntu/)
- [RHEL](https://docs.docker.com/engine/install/rhel/)
- [Rocky Linux](https://docs.docker.com/engine/install/rhel/)
- [AlmaLinux](https://docs.docker.com/engine/install/rhel/)

Astra Linux 1.7 / Debian

Добавьте официальный репозиторий Docker:

```
sudo apt update
sudo apt install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/debian/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/debian \
  $(. /etc/os-release && echo "$VERSION_CODENAME") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt update
```

- `$(dpkg --print-architecture)` — подставит архитектуру системы (обычно `amd64`).
- `$(. /etc/os-release && echo "$VERSION_CODENAME")` — подставит кодовое имя дистрибутива (`bookworm` для Astra Linux 1.7 / Debian 12).

Установите Docker Engine, CLI, containerd и Compose Plugin:

```
sudo apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

Проверьте, что Docker работает:

```
sudo systemctl status docker
```


Astra Linux 1.8

Установите необходимые пакеты для работы репозиториев:

```
sudo apt update
sudo apt install ca-certificates curl
```

Добавьте официальный GPG-ключ Docker:

```
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/debian/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

Добавьте репозиторий Docker. Для Astra Linux 1.8 (Debian 12 Bookworm) используйте:

```
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/debian \
  bookworm stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt update
```

Установите Docker Engine, CLI, containerd и Compose Plugin:

```
sudo apt install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

Проверьте, что Docker работает:

```
sudo systemctl status docker
```


RHEL 9.7

Установите необходимые пакеты для работы репозиториев:

```
sudo dnf install dnf-plugins-core
```

Добавьте официальный репозиторий Docker:

```
sudo dnf config-manager --add-repo https://download.docker.com/linux/rhel/docker-ce.repo
```

Установите Docker Engine, CLI, containerd и Compose Plugin:

```
sudo dnf install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

Запустите Docker:

```
sudo systemctl start docker
```

Проверьте, что Docker работает:

```
sudo systemctl status docker
```


Rocky Linux 9.7

Установите необходимые пакеты для работы репозиториев:

```
sudo dnf install dnf-plugins-core
```

Добавьте официальный репозиторий Docker:

```
sudo dnf config-manager --add-repo https://download.docker.com/linux/rocky/docker-ce.repo
```

Установите Docker Engine, CLI, containerd и Compose Plugin:

```
sudo dnf install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

Запустите Docker:

```
sudo systemctl start docker
```

Проверьте, что Docker работает:

```
sudo systemctl status docker
```


AlmaLinux 9.7

Установите необходимые пакеты для работы репозиториев:

```
sudo dnf install dnf-plugins-core
```

Добавьте официальный репозиторий Docker:

```
sudo dnf config-manager --add-repo https://download.docker.com/linux/rhel/docker-ce.repo
```

Установите Docker Engine, CLI, containerd и Compose Plugin:

```
sudo dnf install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

Запустите Docker:

```
sudo systemctl start docker
```

Проверьте, что Docker работает:

```
sudo systemctl status docker
```


RED OS 8

Установите необходимые пакеты для работы репозиториев:

```
sudo dnf install dnf-plugins-core
```

Добавьте официальный репозиторий Docker:

```
sudo dnf config-manager --add-repo https://download.docker.com/linux/centos/docker-ce.repo
```

Установите Docker Engine, CLI, containerd и Compose Plugin:

```
sudo dnf install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

Запустите Docker:

```
sudo systemctl start docker
```

Проверьте, что Docker работает:

```
sudo systemctl status docker
```

Установка ИК.Node

Установите ИК.Node по инструкции для вашего дистрибутива:

- [Astra Linux](https://docs.alarm.ru/ik-node/install/debian/)
- [Debian](https://docs.alarm.ru/ik-node/install/debian/)
- [Ubuntu](https://docs.alarm.ru/ik-node/install/ubuntu/)
- [RHEL](https://docs.alarm.ru/ik-node/install/rhel/)
- [Rocky Linux](https://docs.alarm.ru/ik-node/install/rhel/)
- [AlmaLinux](https://docs.alarm.ru/ik-node/install/rhel/)
- [RED OS](https://docs.alarm.ru/ik-node/install/rhel/)

Установка ПомойИ

Установите ПомойИ по инструкции для вашего дистрибутива:

- [Astra Linux](https://docs.alarm.ru/pomoyi/install/debian/)
- [Debian](https://docs.alarm.ru/pomoyi/install/debian/)
- [Ubuntu](https://docs.alarm.ru/pomoyi/install/ubuntu/)
- [RHEL](https://docs.alarm.ru/pomoyi/install/rhel/)
- [Rocky Linux](https://docs.alarm.ru/pomoyi/install/rhel/)
- [AlmaLinux](https://docs.alarm.ru/pomoyi/install/rhel/)
- [RED OS](https://docs.alarm.ru/pomoyi/install/rhel/)

Установка и настройка nginx (ОС Astra Linux 1.8)

Установите nginx:

```
sudo apt update
sudo apt install nginx
```

Создайте конфигурацию сайта:

```
sudo nano /etc/nginx/sites-available/alarm
```

Вставьте следующее содержимое:

```
server {
    listen 80;
    server_name _;

    location / {
        proxy_pass http://127.0.0.1:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}
```

Включите сайт и перезапустите nginx:

```
sudo ln -s /etc/nginx/sites-available/alarm /etc/nginx/sites-enabled/
sudo nginx -t
sudo systemctl restart nginx
```

Шаг 2. Установка АЛСМ

Убедитесь, что выполнены все предварительные требования:

- Linux-ядро, glibc и архитектура соответствуют требованиям
- Установлен и запущен Docker Engine с Compose Plugin
- Установлен и настроен nginx

Установите пакет `alarm`:

```
sudo apt install alarm
```

Во время установки будет запрошен пароль для PostgreSQL — задайте надёжный пароль и сохраните его.

Проверьте, что пакет установлен:

```
dpkg -l alarm
```

Ожидаемый вывод:

```
ii  alarm  1.0.0  amd64  ...
```

Шаг 3. Запуск АЛСМ

Перейдите в рабочий каталог и запустите контейнеры:

```
cd /opt/alarm
sudo docker compose up -d
```

Проверьте, что все контейнеры запущены:

```
sudo docker compose ps
```

Ожидаемый вывод — все сервисы в состоянии `Up`:

```
NAME                IMAGE                          COMMAND                  SERVICE             CREATED          STATUS          PORTS
alarm-core-1        registry.alarm.ru/alarm-core   "/usr/local/bin/alar…"   core                2 minutes ago    Up 2 minutes    127.0.0.1:8080->8080/tcp
alarm-postgres-1    postgres:16                    "docker-entrypoint.s…"   postgres            2 minutes ago    Up 2 minutes    5432/tcp
```

Шаг 4. Проверка установки

Проверка статуса Docker-контейнеров

Убедитесь, что все контейнеры работают:

```
sudo docker compose ps
```

Все сервисы должны быть в состоянии `Up`.

Проверка доступности по http и https

Проверьте доступность сайта по протоколу http:

```
curl -I http://localhost
```

Ожидаемый ответ — `HTTP/1.1 200 OK` или `HTTP/1.1 302 Found`:

```
HTTP/1.1 200 OK
Server: nginx/1.24.0
Date: Mon, 01 Jan 2024 12:00:00 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 1234
Connection: keep-alive
```

Проверьте доступность сайта по протоколу https:

```
curl -I -k https://localhost
```

Ключ `-k` отключает проверку сертификата (для самоподписанного сертификата).

Ожидаемый ответ — `HTTP/2 200` или `HTTP/2 302`:

```
HTTP/2 200
server: nginx/1.24.0
date: Mon, 01 Jan 2024 12:00:00 GMT
content-type: text/html; charset=utf-8
content-length: 1234
```

Проверка сайта АЛСМ в браузере

Откройте в браузере адрес `https://<IP-адрес-сервера>` (или `https://localhost`, если браузер запущен на том же сервере).

Браузер покажет предупреждение о недоверенном сертификате — это нормально для самоподписанного сертификата. Нажмите «Дополнительно» → «Перейти на сайт».

Страница входа АЛСМ

Проверка веб-интерфейса АЛСМ

- Откройте страницу и проверьте отображение элементов интерфейса.
- Убедитесь, что нет ошибок в консоли браузера (F12 → Console).
- Попробуйте войти в систему с учётными данными по умолчанию.
- Проверьте работу основных функций.

Рабочий стол АЛСМ

Шаг 5. Обновление АЛСМ

Перед обновлением:

- Сделайте резервную копию базы данных и конфигурации (см. раздел «Резервное копирование»).
- Ознакомьтесь с [журналом изменений](https://docs.alarm.ru/changelog/) — там описаны все изменения между версиями.
- Убедитесь, что у вас достаточно свободного места на диске.

Обновите пакет `alarm`:

```
sudo apt update
sudo apt install alarm
```

Перезапустите контейнеры:

```
cd /opt/alarm
sudo docker compose up -d
```

Проверьте, что все контейнеры работают:

```
sudo docker compose ps
```

Проверьте доступность сайта:

```
curl -I http://localhost
```

Ожидаемый ответ — `HTTP/1.1 200 OK` или `HTTP/1.1 302 Found`.

Откройте в браузере адрес `https://<IP-адрес-сервера>` и убедитесь, что интерфейс загружается.

Если что-то пошло не так, откатитесь на предыдущую версию:

```
sudo apt install alarm=<предыдущая-версия>
cd /opt/alarm
sudo docker compose up -d
```

Список доступных версий можно посмотреть командой `apt-cache policy alarm`.

Подготовка хостов

Назад к содержанию

Перед установкой нового кластера необходимо добавить информацию о его будущих хостах в ADCM. Это задача **хостпровайдеров** — специальных плагинов, отвечающих за взаимодействие между хостами и ADCM.

ADCM поддерживает несколько типов хостпровайдеров. Для получения подробной информации о том, как их настроить и использовать для создания хостов, обратитесь к документации:

- SSH
- Yandex Cloud
- VCloud (VMware)
- Cloud.ru Advanced
- CROC Cloud

Для offline-установки рекомендуется использование SSH-хостпровайдера.

ВНИМАНИЕ

Если ADS или ADS Control планируется использовать совместно с ADPS, при создании хостов ADS/ADS Control необходимо указывать их полные доменные имена (FQDN). Максимальная длина FQDN — 48 символов.

Создание кластера

Дарья Барышева

Содержание

Назад к содержанию

Шаг 1. Получение Enterprise-версии бандла

Шаг 2. Загрузка бандла в ADCM

Шаг 3. Создание кластера на базе загруженного бандла

В ADCM **кластер** означает набор сервисов, запущенных на группе хостов.

Дистрибутивы кластеров для ADCM поставляются в виде бандлов. В общем случае бандл представляет обычный архив, описывающий логику развертывания кластера.

Шаги для добавления нового кластера приведены ниже.

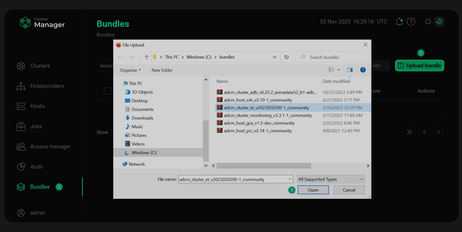
Шаг 1. Получение Enterprise-версии бандла

Пожалуйста, запросите бандл у команды поддержки Arenadata.

Шаг 2. Загрузка бандла в ADCM

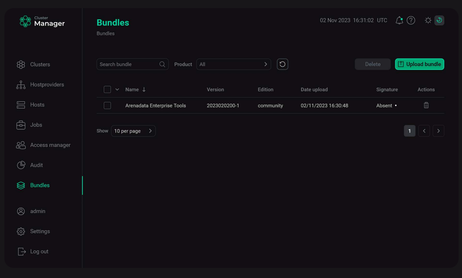
Чтобы загрузить бандл в ADCM:

1. Выберите пункт ***Bundles*** в левом навигационном меню и нажмите ***Upload bundle***.
2. Выберите бандл в открывшемся диалоговом окне.



Переход к загрузке бандла

3. После успешной загрузки бандл отображается на странице ***Bundles***.



Результат успешной загрузки бандла

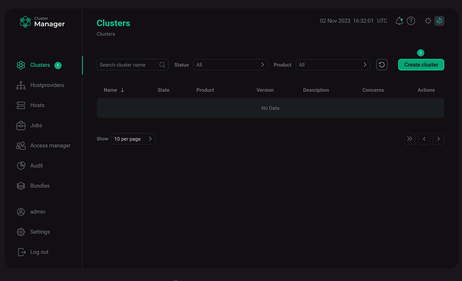
Шаг 3. Создание кластера на базе загруженного бандла

ВАЖНО

На этапе создания кластера через web-интерфейс ADCM генерируется только сам кластер. Это означает добавление записи о кластере в ADCM — сервисы на этом шаге еще не устанавливаются.

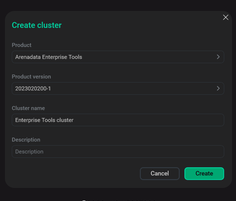
Для создания кластера необходимо выполнить шаги:

1. Выберите пункт ***Clusters*** в левом навигационном меню и нажмите ***Create cluster***.



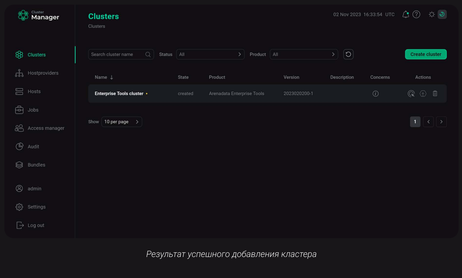
Переход к созданию кластера

2. В открывшемся окне требуется описать новый кластер:
   - Выберите загруженный бандл в поле ***Product***.
   - Выберите версию бандла в поле ***Product version***. Несколько версий становятся доступны после загрузки нескольких бандлов для одного типа продуктов.
   - Введите имя кластера в поле ***Cluster name***.
   - При необходимости добавьте описание кластера в поле ***Description***.
   - Установите флаг ***I accept Terms of Agreement*** для подписания лицензионного соглашения. Флаг отображается только для бандлов версии Enterprise. Чтобы ознакомиться с текстом соглашения, перейдите по ссылке ***Terms of Agreement***. Подписать соглашение также возможно на странице ***Bundles***, нажав на соответствующий продуктовый бандл.
   - Нажмите ***Create***.



Создание кластера

3. После успешного добавления кластер отображается на странице ***Clusters***.



Результат успешного добавления кластера

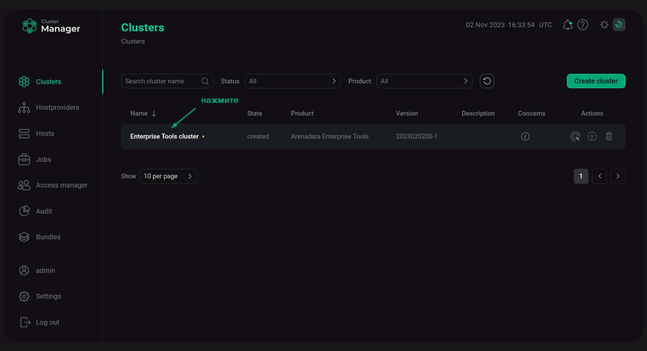
Добавление сервисов

Дарья Барышева

Назад к содержанию

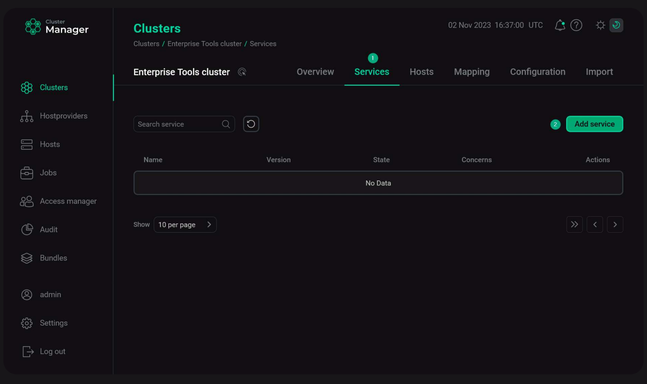
В ADCM **сервис** означает программное обеспечение, выполняющее некоторую функцию. Для добавления сервисов в кластер:

1. Выберите кластер на странице ***Clusters***. Для этого нажмите на имя кластера в столбце ***Name***.



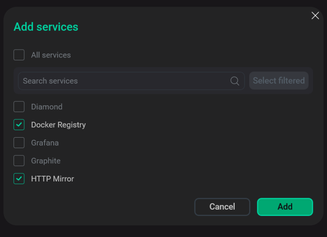
Выбор кластера

2. Откройте вкладку ***Services*** на странице кластера и нажмите ***Add service***.



Переход к добавлению сервисов

3. В открывшемся окне выберите сервисы для добавления в кластер и нажмите ***Add***.



Выбор сервисов

Краткое описание доступных сервисов приведено ниже.

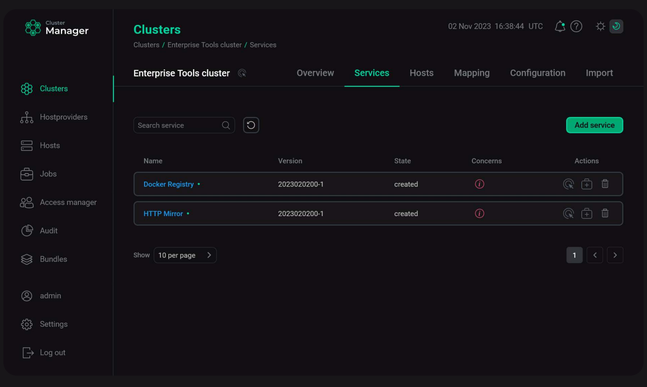
Сервисы, которые необходимо добавить в кластер Enterprise Tools

Сервис	Назначение
HTTP Mirror	Сервис, используемый для размещения пакетов репозиториев, необходимых для offline-установки продуктов
Docker Registry	Сервис, используемый для хранения образов Docker, необходимых для offline-установки продуктов

ПРИМЕЧАНИЕ

Сервис CoreDNS не используется (deprecated). Сервисы Diamond, Graphite и Grafana необходимы для настройки мониторинга — они будут описаны отдельно.

4. В результате успешно добавленные сервисы отображаются на вкладке ***Services***.



Результат успешного добавления сервисов в кластер

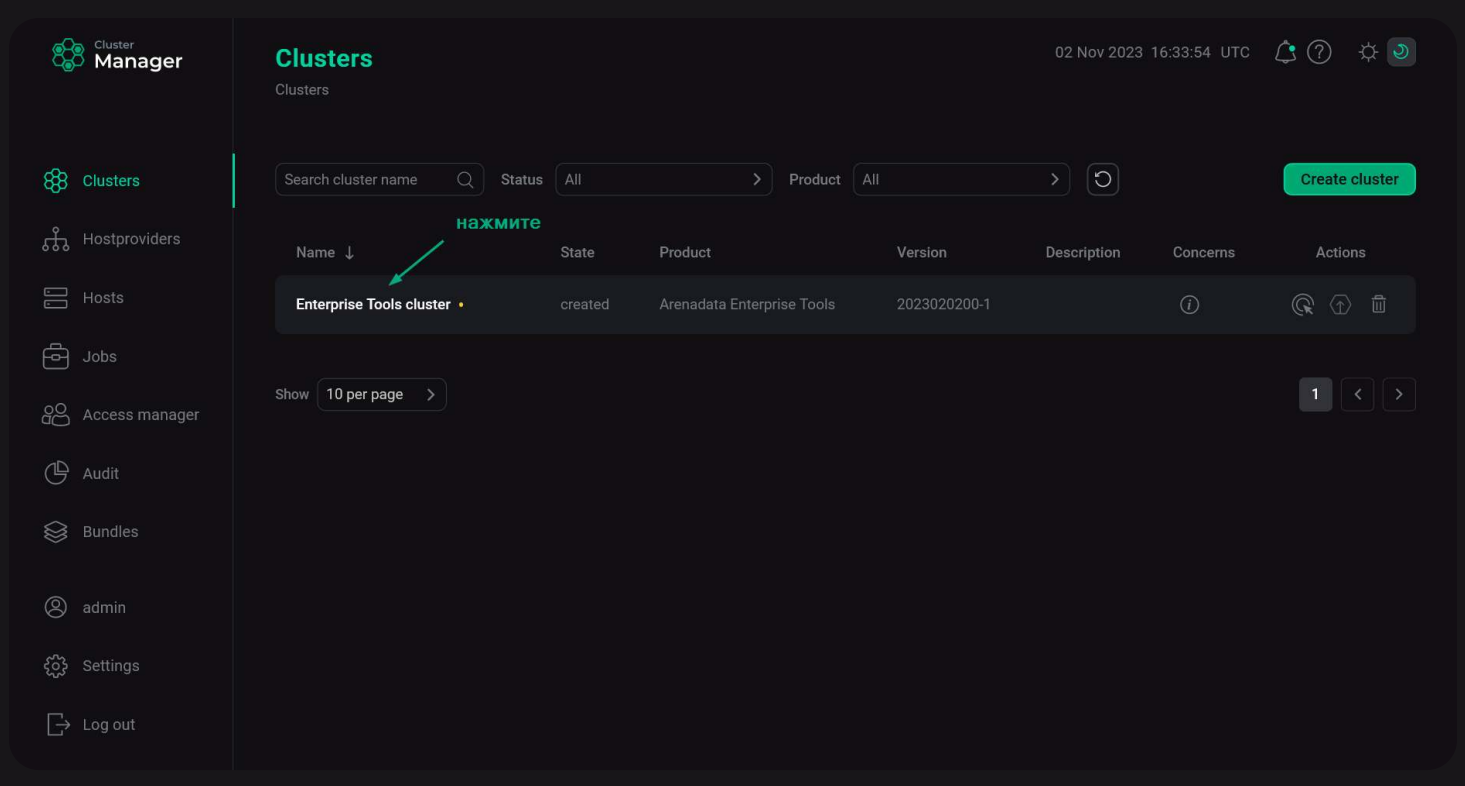
Добавление хостов в кластер

Дарья Барышева

Назад к содержанию

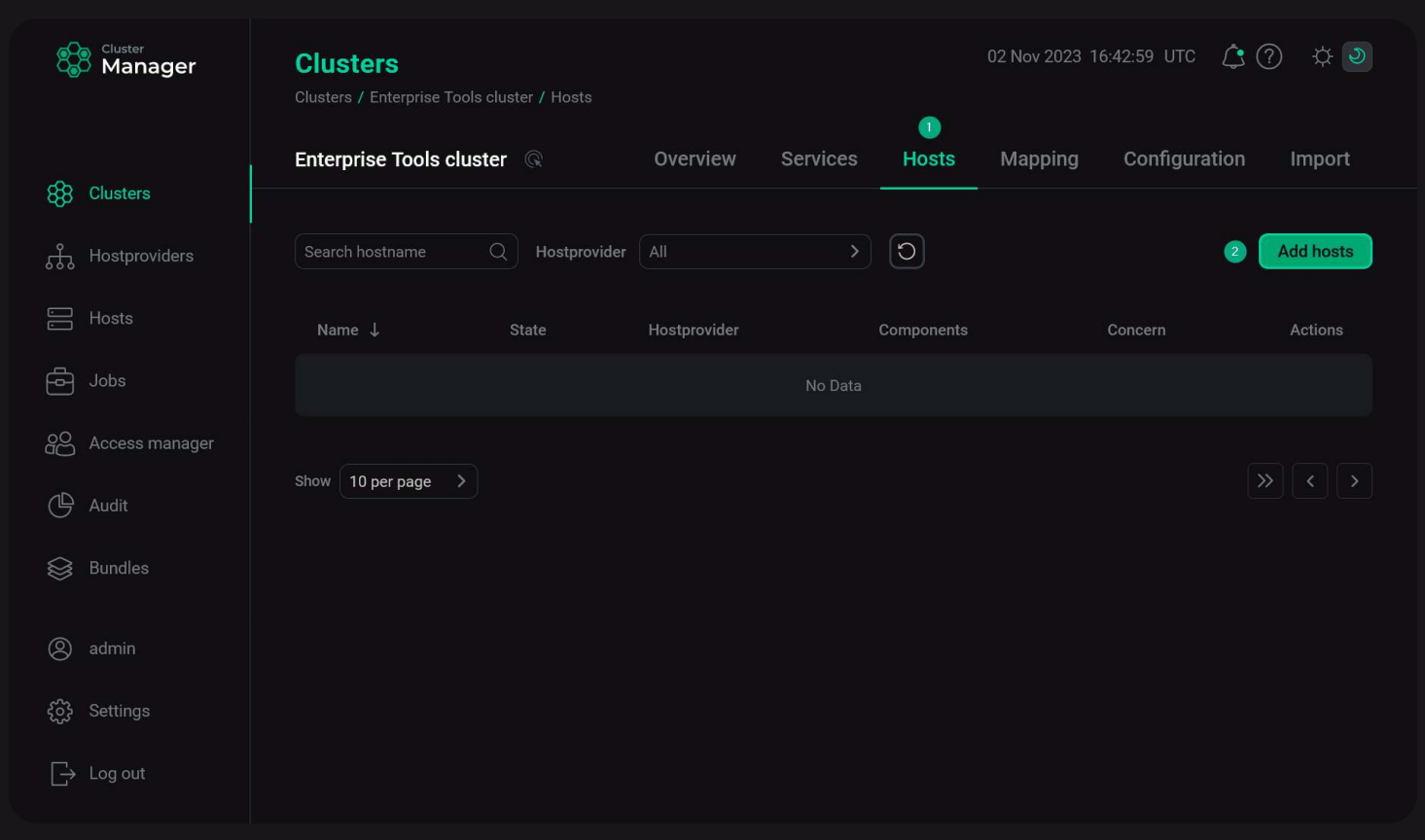
Для добавления хостов в кластер:

1. Выберите кластер на странице ***Clusters***. Для этого нажмите на имя кластера в столбце ***Name***.



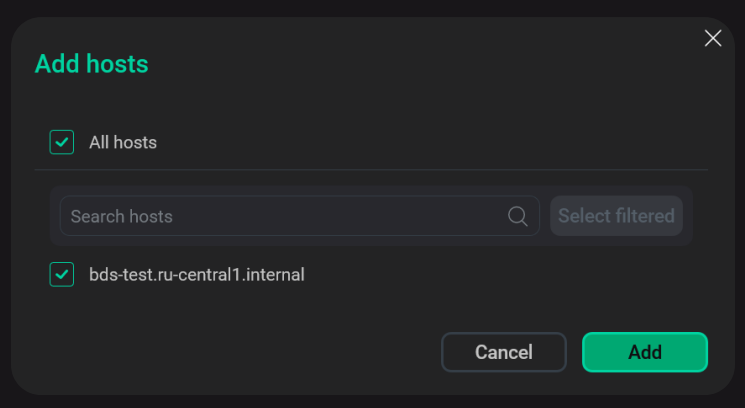
Выбор кластера

2. Откройте вкладку ***Hosts*** на странице кластера и нажмите ***Add hosts***.



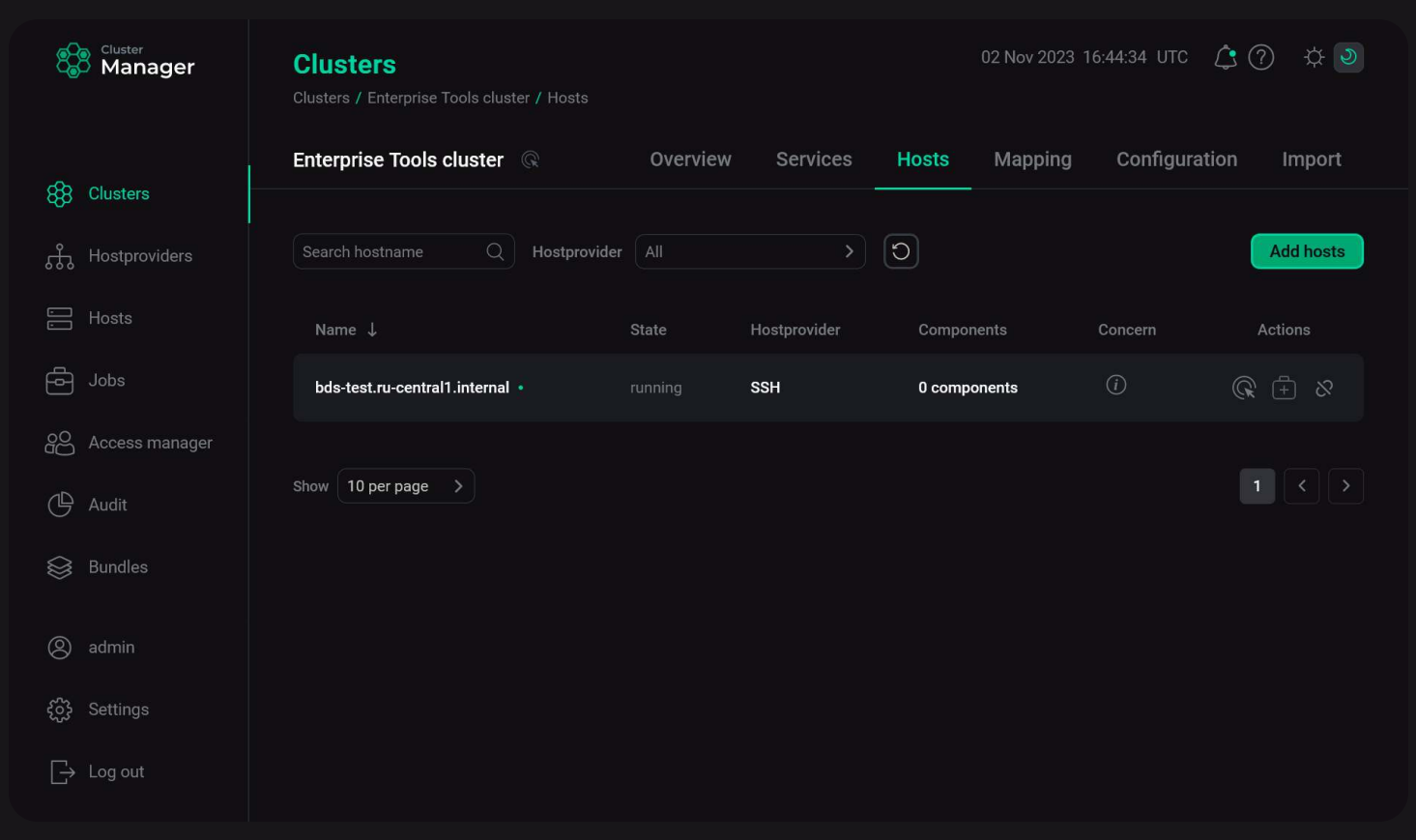
Переход к добавлению хостов

3. В открывшемся окне выберите хосты для добавления в кластер и нажмите ***Add***.



Выбор хостов

4. В результате успешно добавленные хосты отображаются на вкладке ***Hosts***.



Результат успешного добавления хостов в кластер

Добавление компонентов

Дарья Барышева

Назад к содержанию

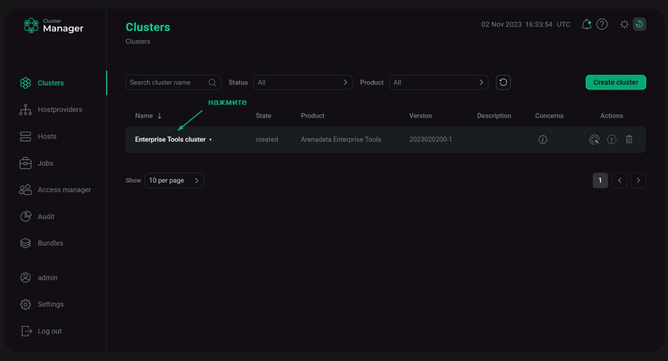
В ADCM **компонент** означает часть сервиса, которая должна быть развернута на одном или нескольких хостах кластера.

ВАЖНО

Изначально на хостах кластера нет ни одного компонента — распределение компонентов между хостами должно быть проведено вручную.

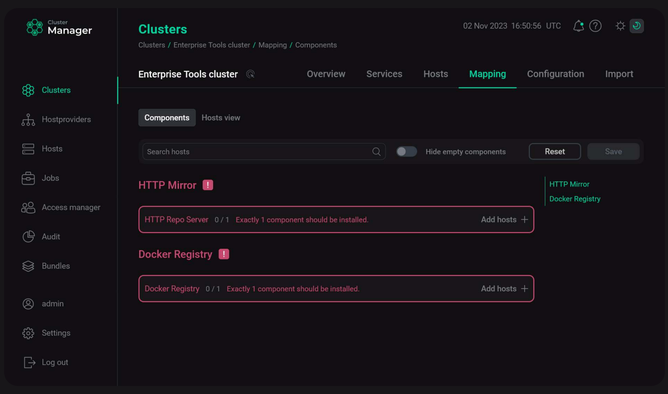
Для добавления компонентов на хосты кластера:

1. Выберите кластер на странице ***Clusters***. Для этого нажмите на имя кластера в столбце ***Name***.



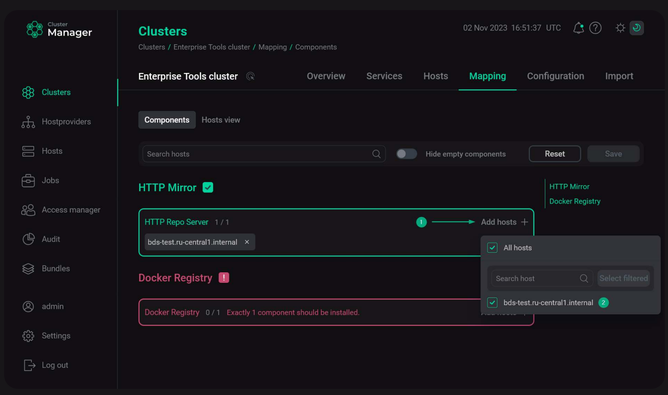
Выбор кластера

2. Откройте вкладку ***Mapping*** на странице кластера.



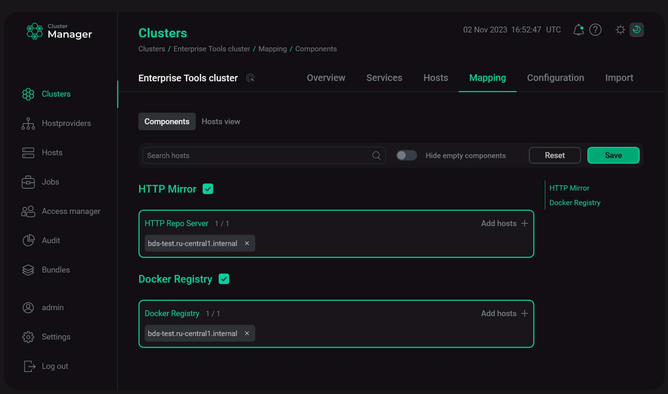
Переход к распределению сервисных компонентов

3. Распределите все компоненты между хостами кластера. Для назначения хостов компоненту нажмите ***Add hosts*** и выберите хосты в открывшемся окне.



Выбор хостов для компонента

4. После завершения распределения компонентов нажмите ***Save***.



Сохранение распределения компонентов

Установка кластера

Ольга Семме, Дарья Барышева

Содержание

Назад к содержанию

Шаг 1. Запуск установки

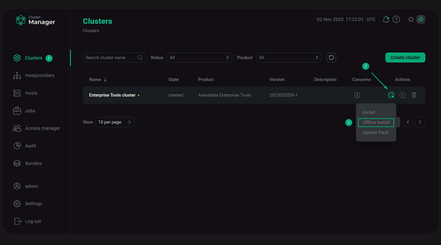
Шаг 2. Загрузка пакетов для offline-установки

Шаг 3. Проверка результатов

Шаг 1. Запуск установки

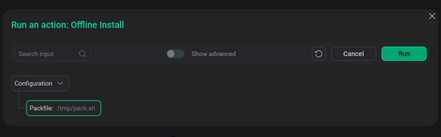
Для установки кластера Enterprise Tools выполните следующие шаги:

1. Запустите действие ***Offline Install*** для кластера, выбранного на странице ***Clusters***, нажав на иконку в столбце ***Actions***.



Переход к установке кластера

2. В открывшемся диалоговом окне введите полный путь к shell-скрипту, полученному от команды поддержки Arenadata, и нажмите ***Run***.

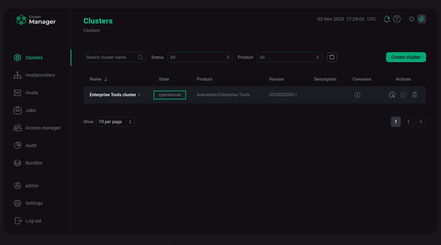


Подтверждение установки

3. Дождитесь завершения процесса установки.

ПРИМЕЧАНИЕ

В случае ошибок логи можно найти на странице ***Jobs***.

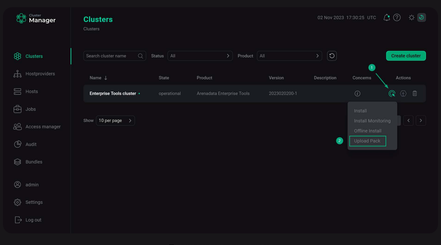


Процесс установки завершен

Шаг 2. Загрузка пакетов для offline-установки

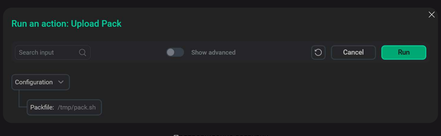
В дополнение к установке сервисов ET требуется загрузка всех установочных файлов и образов в инициализированный http-репозиторий и Docker Registry на хосте, где развернут кластер ET. Это важно для последующего offline-развертывания продуктов Arenadata. Для загрузки выполните шаги:

1. Запустите действие ***Upload Pack*** для кластера, выбранного на странице ***Clusters***, нажав на иконку в столбце ***Actions***.



Переход к загрузке пакетов

2. В открывшемся диалоговом окне введите полный путь к shell-скрипту, полученному от команды поддержки Arenadata, и нажмите ***Run***.



Подтверждение загрузки

3. Дождитесь завершения процесса загрузки.

ПРИМЕЧАНИЕ

В случае ошибок логи можно найти на странице ***Jobs***.

Шаг 3. Проверка результатов

Для проверки установки на хосте, где развернут кластер, выполните шаги:

1. Убедитесь, что порты `443` и `81` находятся в статусе `LISTEN`. Первый порт используется сервисом Docker Registry, второй порт — сервисом HTTP Mirror.

```
$ sudo netstat -ntlp|grep L|grep -E "81|443"
```

Результат команды:

```
tcp6       0      0 :::443                  :::*                    LISTEN      1694/docker-proxy-c
tcp6       0      0 :::81                   :::*                    LISTEN      1708/docker-proxy-c
```

2. Проверьте наличие директории */opt/arenadata/etools/httprepo/arenadata-repo*. Она должна содержать файлы, необходимые для offline-установки продуктов, выбранных вами на этапе получения shell-скрипта от команды поддержки Arenadata.

```
$ ls -la /opt/arenadata/etools/httprepo/arenadata-repo
```

Следующий пример показывает данные, которые могут храниться в директории в случае установки ADS и ADS Control .

```
total 0
drwxr-xr-x. 5 root root 45 May 23 16:01 .
drwxr-xr-x. 3 root root 28 May 23 16:00 ..
drwxr-xr-x. 3 root root 17 May 23 16:01 ADM
drwxr-xr-x. 3 root root 19 May 23 16:00 ADS
drwxr-xr-x. 3 root root 21 Nov 23 19:12 ADSCC
drwxr-xr-x. 3 root root 19 May 23 16:01 zookeeper
```

Создание кластера

Ольга Семье, Дарья Барышева

Содержание

Назад к содержанию

Шаг 1. Получение Enterprise-версии бандла
Шаг 2. Загрузка бандла в ADCM
Шаг 3. Создание кластера на базе загруженного бандла

В ADCM **кластер** означает набор сервисов, запущенных на группе хостов.

Дистрибутивы кластеров для ADCM поставляются в виде бандлов. В общем случае бандл представляет обычный архив, описывающий логику развертывания кластера.

Шаги для добавления нового кластера приведены ниже.

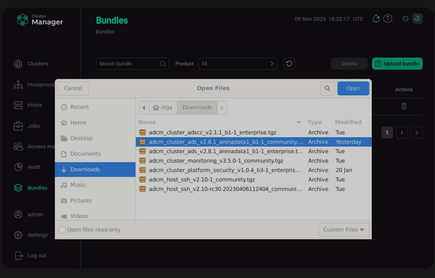
Шаг 1. Получение Enterprise-версии бандла

Пожалуйста, запросите бандл у команды поддержки Arenadata.

Шаг 2. Загрузка бандла в ADCM

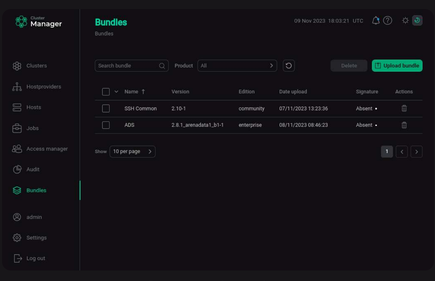
Чтобы загрузить бандл в ADCM:

1. Выберите пункт ***Bundles*** в левом навигационном меню и нажмите ***Upload bundle***.
2. Выберите бандл в открывшемся диалоговом окне.



Переход к загрузке бандла

3. После успешной загрузки бандл отображается на странице ***Bundles***.



Результат успешной загрузки бандла

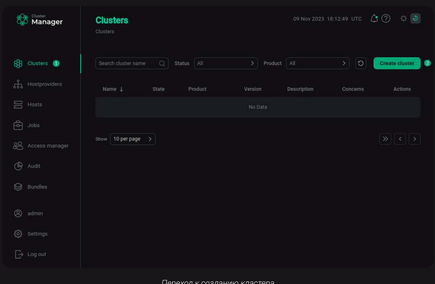
Шаг 3. Создание кластера на базе загруженного бандла

ВАЖНО

На этапе создания кластера через web-интерфейс ADCM генерируется только сам кластер. Это означает добавление записи о кластере в ADCM — сервисы на этом шаге еще не устанавливаются.

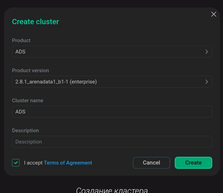
Для создания кластера необходимо выполнить шаги:

1. Выберите пункт ***Clusters*** в левом навигационном меню и нажмите ***Create cluster***.



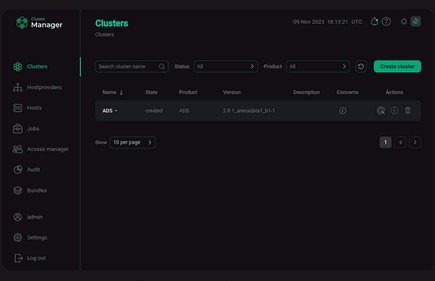
Переход к созданию кластера

2. В открывшемся окне требуется описать новый кластер:
   - Выберите загруженный бандл в поле ***Product***.
   - Выберите версию бандла в поле ***Product version***. Несколько версий становятся доступны после загрузки нескольких бандлов для одного типа продуктов.
   - Введите имя кластера в поле ***Cluster name***.
   - При необходимости добавьте описание кластера в поле ***Description***.
   - Установите флаг ***I accept Terms of Agreement*** для подписания лицензионного соглашения. Флаг отображается только для бандлов версии Enterprise. Чтобы ознакомиться с текстом соглашения, перейдите по ссылке ***Terms of Agreement***. Подписать соглашение также возможно на странице ***Bundles***, нажав на соответствующий продуктовый бандл.
   - Нажмите ***Create***.



Создание кластера

3. После успешного добавления кластер отображается на странице ***Clusters***.



Результат успешного добавления кластера

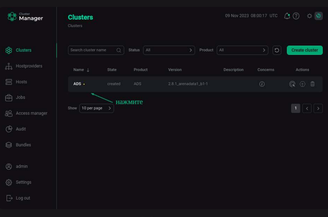
Добавление сервисов

Ольга Семак, Дарья Барышева

Назад к содержанию

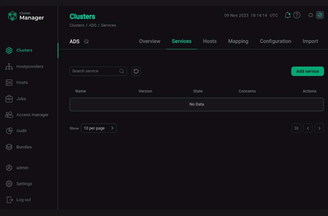
В ADCM **сервис** означает программное обеспечение, выполняющее некоторую функцию. Примеры сервисов в кластерах ADS: NiFi, Kafka, ZooKeeper и другие. Для добавления сервисов в кластер:

1. Выберите кластер на странице **Clusters**. Для этого нажмите на имя кластера в столбце **Name**.



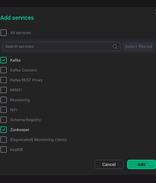
Выбор кластера

2. Откройте вкладку **Services** на странице кластера и нажмите ***Add service***.



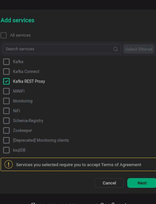
Переход к добавлению сервисов

3. В открывшемся окне выберите сервисы для добавления в кластер и нажмите ***Add***.



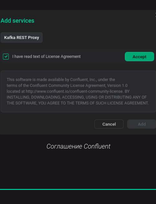
Выбор сервисов

При добавлении сервисов ksqlDB, Schema-Registry, Kafka REST Proxy требуется подписать лицензионное соглашение Confluent. Для этого перейдите к соглашению, нажав ***Next***.



Переход к соглашению Confluent

Далее ознакомьтесь с текстом соглашения и подпишите его, нажав ***Accept***.



Соглашение Confluent

Нажмите ***Add*** для добавления сервиса.

Краткое описание доступных сервисов приведено ниже.

Сервисы, которые могут быть добавлены в кластер ADS

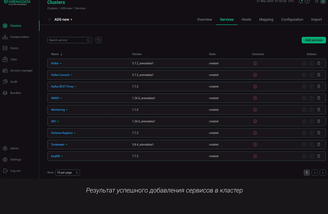
Сервис	Назначение
ZooKeeper	Сервис централизованной координации распределенных приложений, используется для хранения метаданных о разделах (partitions) топиков (topics) и брокеров Apache Kafka, а также для выбора брокера в качестве контроллера Kafka. Сообщает брокерам Kafka о текущем состоянии кластера
Kafka	Распределенная платформа потоковой передачи событий с открытым исходным кодом, используемая для высокопроизводительных конвейеров данных, потоковой аналитики, интеграции данных и критически важных приложений
Kafka REST Proxy	Сервис предоставляет RESTful-интерфейс для кластера Kafka, что упрощает создание и прием сообщений, просмотр состояния кластера и выполнение административных действий без использования собственного протокола Kafka или клиентов
Schema-registry	Сервис выполняет функции: • Хранение версионной истории всех схем Avro на основе указанной стратегии имен субъектов с поддержкой нескольких параметров совместимости. • Возможность изменения схем Avro в соответствии с настроенными параметрами совместимости и расширенной поддержкой этих типов схем. • Обеспечение интерфейса RESTful для хранения и извлечения схем Avro. • Предоставление сериализаторов, которые подключаются к клиентам Apache Kafka, обрабатывающим хранение и извлечение схемы сообщений.
ksqlDB	Сервис представляет собой потоковый SQL-движок с открытым исходным кодом, который обеспечивает обработку данных в реальном времени с использованием Apache Kafka
NiFi	Распределенная платформа обработки событий (сообщений), обеспечивающая управление потоками данных из различных источников в режиме реального времени с помощью графического интерфейса
NiFi2	Сервис NiFi с версией 2.x. Сервис добавлен в режиме технологического превью и не предназначен для использования в производственных средах. Существующие ограничения для использования NiFi2: • NiFi2 может быть установлен только в составе отдельного кластера (NiFi2 + Zookeeper). • Перед установкой сервиса необходимо включить установку Java из репозиториев arenadata, используя параметр ***arenadata_java*** репозитория используемой ОС для установки Java 21 при настройке кластера.
MiNiFi	Подпроект Apache NiFi, представляющий собой дополнительный подход к сбору данных, который дополняет основные принципы NiFi в управлении потоками данных, фокусируясь на сборе данных в источнике их создания
Monitoring	Сервис разворачивает в ADS свой сервер Prometheus, который реализует сбор и хранение метрик мониторинга кластера ADS, а также поддерживает возможность использования веб-приложения Grafana для визуализации и анализа информации
Monitoring Clients	Агенты, отправляющие информацию о хосте и сервисах в мониторинг. Сервис необходимо добавить после установки и настройки кластера мониторинга, если планируется мониторинг текущего кластера
Kafka Connect	Инструмент для масштабируемой и надежной потоковой передачи данных между Kafka и другими системами данных

Минимальный набор сервисов, рекомендуемый для кластеров ADS, приведен ниже:

- NiFi
- Kafka
- ZooKeeper

Эти сервисы составляют основу Streaming, и их достаточно для быстрой и простой установки и управления потоковой передачей с помощью Arenadata Cluster Manager. Полный перечень сервисов зависит от требований конкретного проекта. Могут быть установлены не все сервисы. Например, если вы не планируете использовать NiFi, то добавлять сервис не требуется. Если использовать сервис мониторинга (не на основе Graphite), нет необходимости устанавливать агенты из клиентов мониторинга. Если вы планируете использовать Kafka, вам потребуются службы Kafka и ZooKeeper. Сервис может состоять из обязательных и необязательных компонентов. Например, сервис ksqlDB состоит из обязательного серверного компонента и необязательного клиентского компонента.

4. В результате успешно добавленные сервисы отобразятся на вкладке **Services**.



Результат успешного добавления сервисов в кластер

ПРИМЕЧАНИЕ

Опциональные сервисы могут быть добавлены в кластер позднее. Процесс добавления сервисов в уже развернутый кластер не отличается от описанного выше.

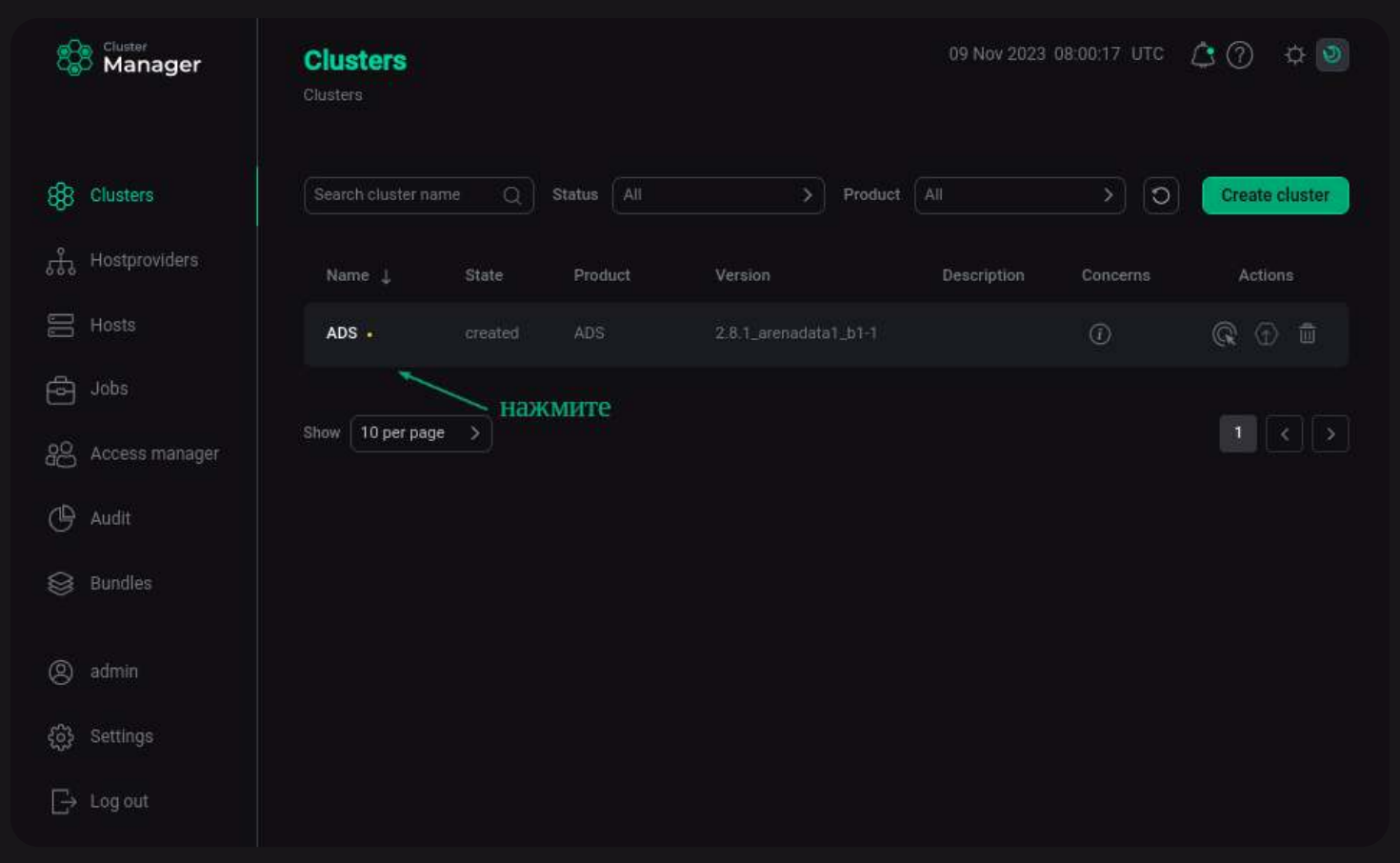
Добавление хостов в кластер

Ольга Семме, Дарья Барышева

Назад к содержанию

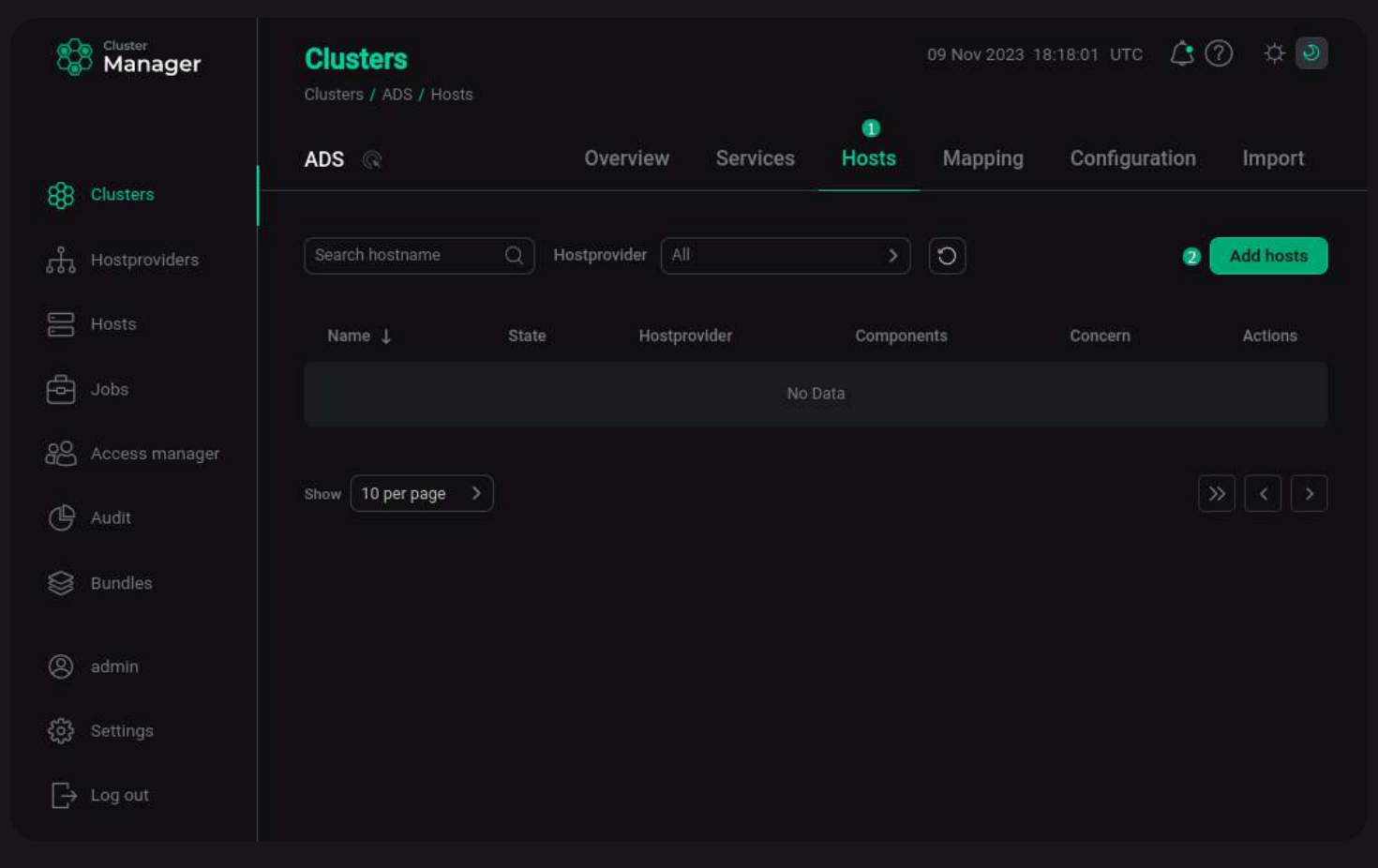
Для добавления хостов в кластер:

1. Выберите кластер на странице ***Clusters***. Для этого нажмите на имя кластера в столбце ***Name***.



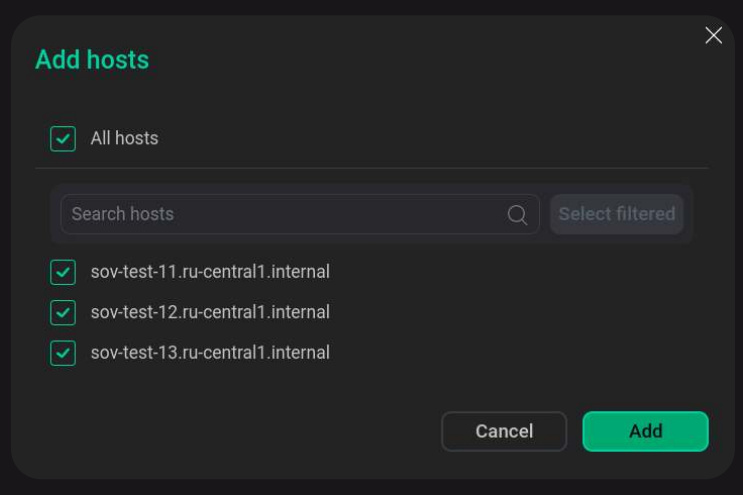
Выбор кластера

2. Откройте вкладку ***Hosts*** на странице кластера и нажмите ***Add hosts***.



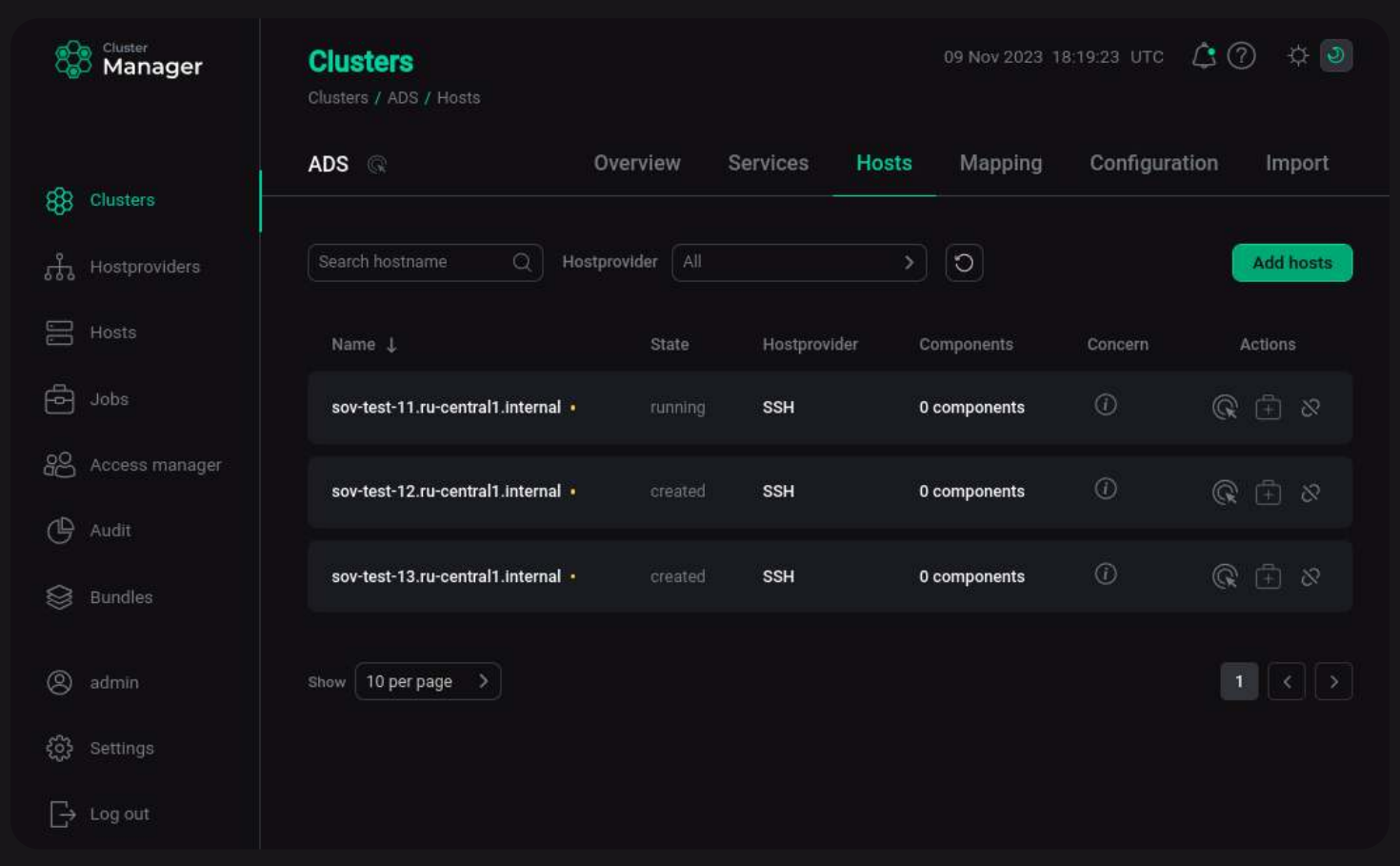
Переход к добавлению хостов

3. В открывшемся окне выберите хосты для добавления в кластер и нажмите ***Add***.



Выбор хостов

4. В результате успешно добавленные хосты отображаются на вкладке ***Hosts***.



Результат успешного добавления хостов в кластер

Добавление компонентов

Ольга Семина, Дарья Барышева

Назад к содержанию

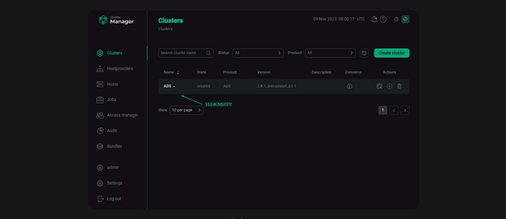
В ADCM **компонент** означает часть сервиса, которая должна быть развернута на одном или нескольких хостах кластера. Например, сервис NiFi состоит из таких компонентов, как NiFi Server и NiFi Registry.

ВАЖНО

Изначально на хостах кластера нет ни одного компонента — распределение компонентов между хостами должно быть проведено вручную.

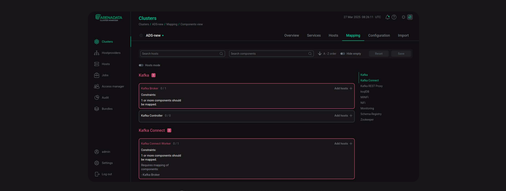
Для добавления компонентов на хосты кластера:

1. Выберите кластер на странице *Clusters*. Для этого нажмите на имя кластера в столбце *Name*.



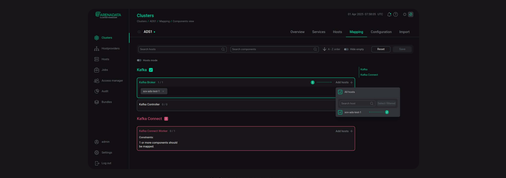
Выбор кластера

2. Откройте вкладку *Mapping* на странице кластера.



Переход к распределению сервисных компонентов

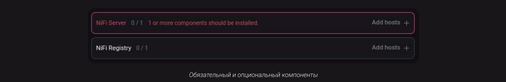
3. Распределите все компоненты между хостами кластера. Для назначения хостов компоненту нажмите *Add hosts* и выберите хосты в открывшемся окне.



Выбор хостов для компонента

Обязательные и опциональные компоненты

Каждый сервис может иметь обязательные и опциональные компоненты. Обязательные подсвечиваются красным цветом. Также в интерфейсе ADCM для них выводится информация о том, в каком количестве необходимо установить каждый компонент.



Обязательный и опциональный компоненты

ВНИМАНИЕ

Без назначения нужного числа хостов обязательным компонентам сохранение общей схемы распределения невозможно.

Следующая таблица содержит примеры распределения компонентов в кластерах ADS, состоящих из 3 и 5 хостов.

Примеры распределения компонентов

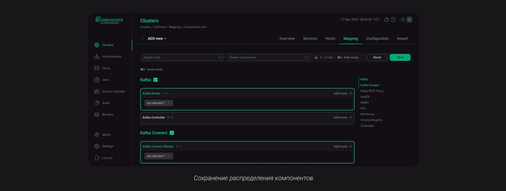
3 хоста

Хост	Сервис	Компонент
Host 1	Zookeeper	Zookeeper server
	Kafka	Kafka Broker
	Kafka REST Proxy	Kafka REST Proxy
	Schema-Registry	Schema-Registry
Host 2	Zookeeper	Zookeeper Server
	ksqlDB	ksqlDB Client ksqlDB Server
	Kafka	Kafka Broker
	NiFi	NiFi Server
Host 3	Zookeeper	Zookeeper Server
	Kafka	Kafka Broker
	Kafka Connect	Kafka Connect Worker
	NiFi	NiFi Server Nifi Registry
	MiNiFi	MiNiFi C2 Server MiNiFi Agent

5 хостов

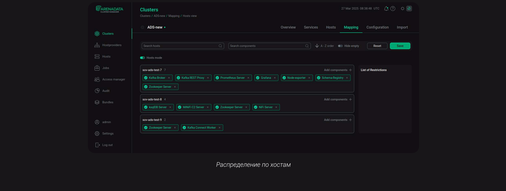
Хост	Сервис	Компонент
Host 1	Zookeeper	Zookeeper Server
	Kafka	Kafka Broker
	Schema-Registry	Schema-Registry
Host 2	Zookeeper	Zookeeper Server
	Kafka	Kafka Broker
Host 3	Zookeeper	Zookeeper Server
	ksqlDB	ksqlDB Client ksqlDB Server
	Kafka REST Proxy	Kafka REST Proxy
Host 4	Kafka	Kafka Broker
	NiFi	NiFi Server NiFi Registry
Host 5	NiFi	NiFi Server
	MiNiFi	MiNiFi C2 Server MiNiFi Agent
	Kafka Connect	Kafka Connect Worker

4. После завершения распределения компонентов нажмите *Save*.



Сохранение распределения компонентов

Начиная с версии 2.3.0 ADCM поддерживает альтернативный способ сопоставления – по хостам. Для этого переведите в активное положение переключатель *Hosts mode* и распределите компоненты между хостами путем нажатия на кнопку *Add components* напротив каждого хоста.



Распределение по хостам

Настройка сервисов

Часть статьи «Дата-платформа»

ADS

Содержание

Назад к содержанию

Конфигурация

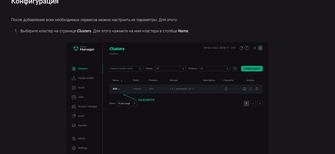
Компоненты

Группы конфигурации

Конфигурация

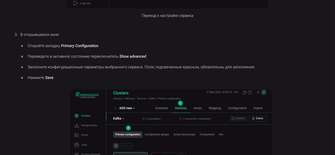
После добавления всех необходимых сервисов можно настроить их параметры. Для этого:

1. Выберите кластер на странице **Clusters**. Для этого нажмите на имя кластера в столбце **Name**



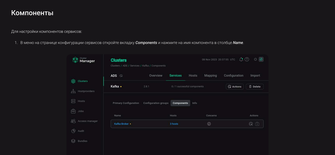
Выбор кластера

2. Откройте вкладку **Services** на странице кластера. Нажмите на имя сервиса в столбце **Name**.

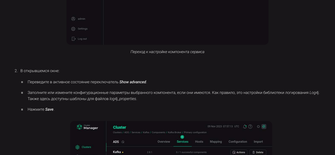


Переход к настройке сервиса

3. В открывшемся окне:
   - Откройте вкладку **Primary Configuration**
   - Переведите в активное состояние переключатель **Show advanced**
   - Заполните конфигурационные параметры выбранного сервиса. Поля, подсвечены красным, обязательны для заполнения
   - Нажмите **Save**



Настройка сервиса



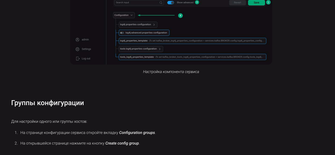
ПРИМЕЧАНИЕ

Более подробную информацию по всем конфигурационным параметрам можно получить в статье Конфигурационные параметры ADS.

Компоненты

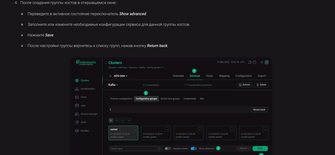
Для настройки компонентов сервисов:

1. В меню на странице кластера сервисов откройте вкладку **Components** и нажмите на имя компонента в столбце **Name**



Переход к настройке компонента сервиса

2. В открывшемся окне:
   - Переведите в активное состояние переключатель **Show advanced**
   - Заполните или измените конфигурационные параметры выбранного компонента, если они имеются. Как правило, это настройки библиотеки log4j2. Также здесь доступны шаблоны для файлов kpdl, properties
   - Нажмите **Save**

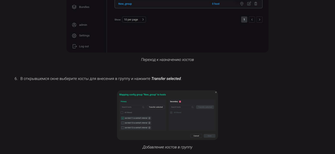


Настройка компонента сервиса

Группы конфигурации

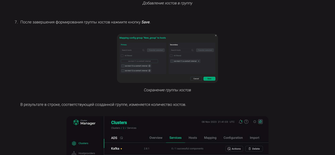
Для настройки одного или группы хостов:

1. На странице конфигурации сервиса откройте вкладку **Configuration groups**
2. На открывшейся странице нажмите на кнопку **Create config group**



Переход к созданию группы хостов

3. В открывшемся окне введите наименование группы и нажмите **Next**



Создание группы хостов

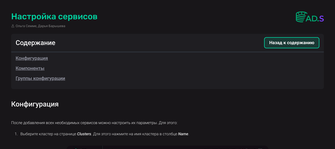
4. После создания группы хостов в открывшемся окне:
   - Переведите в активное состояние переключатель **Show advanced**
   - Заполните или измените необходимые конфигурации сервиса для данной группы хостов
   - Нажмите **Save**
   - После настройки группы вернитесь в списку групп, нажав кнопку **Return back**

Настройка группы хостов

5. Для назначения хостов в группу в старом, соответствующей созданной группе, нажмите на иконку

Переход к назначению хостов

6. В открывшемся окне выберите хосты для внесения в группу и нажмите **Transfer selected**

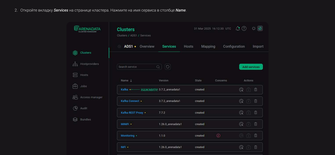


Добавление хостов в группу

7. После завершения формирования группы хостов нажмите кнопку **Save**

Сохранение группы хостов

В результате в группе, соответствующей созданной группе, изменится количество хостов.



Сформированная группа хостов

ПРИМЕЧАНИЕ

Более подробную информацию о настройке сервисов для группы хостов можно получить в статье Настройка конфиг-групп.

Настройка кластера

Ольга Семме, Дарья Барышева

Содержание

Назад к содержанию

Primary configuration

Configuration groups

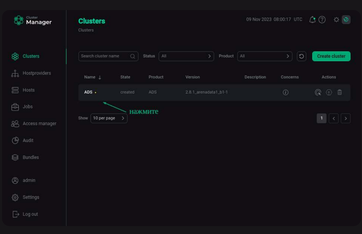
Ansible settings

ВАЖНО

В большинстве случаев кастомная настройка кластера не требуется — значения параметров можно оставить по умолчанию.

После добавления нового кластера можно выполнить его настройку. Для этого выполните шаги:

1. Выберите кластер на странице ***Clusters***. Для этого нажмите на имя кластера в столбце ***Name***.



Выбор кластера

2. Откройте вкладку ***Configuration*** на странице кластера. Вкладка ***Configuration*** состоит из трех подразделов: ***Primary configuration***, ***Configuration groups***, ***Ansible settings***.

3. Заполните необходимые конфигурационные параметры на нужной вкладке и нажмите ***Save***.

Primary configuration

Вкладка ***Configuration → Primary configuration*** содержит основные конфигурационные параметры кластера.

Конфигурационные параметры кластера перечислены ниже:

- ***Reliability control*** – параметры этой секции предназначены для настройки тайм-аута ожидания разблокировки пакетного менеджера DNF. Отображается, если активирован переключатель ***Show advanced*** в верхней части окна. В Ubuntu после перезагрузки хоста запускается обновление, что приводит к блокировке DNF.

Имя	Описание	Значение по умолчанию
retries	Максимальное число повторных попыток	80
delay	Тайм-аут между повторными попытками (в секундах)	30
timeout	Тайм-аут проверки сетевых портов (в секундах)	7200

- Пути к репозиториям, которые будут использованы в процессе установки в зависимости от операционной системы кластера (CentOS 7, Red Hat 7, Ubuntu 22, ALT Linux 8, Astra Linux, RedOS 7). На серверах кластера регистрируются только отмеченные репозитории.

Репозитории

Название репозитория	Параметр	Описание	Значение по умолчанию
monitoring	desc	Описание репозитория	Monitoring components url
	url	Путь к репозиторию Arenadata	Указано в зависимости от ОС
	enabled	Включает использование репозитория	true
ADS	desc	Описание репозитория	ADS repository
	url	Путь к репозиторию Arenadata	Указано в зависимости от ОС
	enabled	Включает использование репозитория	true
prometheus_repo	desc	Описание репозитория	Prometheus repository
	url	Путь к репозиторию Arenadata	Указано в зависимости от ОС
	enabled	Включает использование репозитория	true
arenadata_java	desc	Описание репозитория	Arenadata repository for OpenJDK
	url	Путь к репозиторию Arenadata	Указано в зависимости от ОС
	enabled	Включает использование репозитория. Если установить значение `true` перед действием ***Install***, ***Add/Remove components*** или ***Update*** для кластера или сервиса, на хостах будет установлена последняя версия Java 17	false
ranger plugins	desc	Описание репозитория	Arenadata Ranger plugins repository
	url	Путь к репозиторию Arenadata	Указано в зависимости от ОС
	enabled	Включает использование репозитория	true

- ***Precheck packages*** – включает проверку доступности пакета перед установкой.
- ***Docker registry url*** – указывает размещение Arenadata Docker Registry. Отображается, если активирован переключатель ***Show advanced*** в верхней части окна.
- ***Znode path for cluster in Zookeeper service*** – путь Znode для кластера ADS в сервисе Zookeeper. Отображается, если активирован переключатель ***Show advanced*** в верхней части окна.
- ***BASIC Authentication*** – конфигурация для аутентификации BASIC.
  Описание включения аутентификации приведено в статье Basic-аутентификация.
- ***SASL_PLAINTEXT Authentication*** – конфигурация для аутентификации SASL PLAINTEXT.
  Описание включения аутентификации приведено в статье SASL_PLAINTEXT.
- ***Kerberos*** – конфигурация для аутентификации по протоколу Kerberos.
  Описание включения аутентификации приведено в статьях:
  - MIT Kerberos
  - Kerberos с MS Active Directory
  - Kerberos с Samba
- ***java.io.tmpdir*** – системное свойство Java, которое определяет, куда JVM записывает временные файлы.
- ***Custom JDK settings*** – переключатель, определяющий необходимость использования кастомного пути к Java для сервисов кластера ADS. При активации переключателя становится доступно для заполнения поле ***JAVA_HOME***. В нем требуется указать путь к работающей Java. В выбранном пути должна содержаться папка *bin* с бинарным файлом Java, например: */usr/lib/jvm/jre-openjdk/*. При пустом значении поля ***JAVA_HOME*** для поиска Java используется системный путь по умолчанию: */usr/lib/jvm/java-1.17.0-openjdk.amd64/* (для всех операционных систем, кроме Centos, RedHat и AstraLinux).
- ***Cluster UUID*** – отображает универсальный уникальный идентификатор кластера (universally unique identifier).

Конфигурирование кластера

Configuration groups

Вкладка ***Configuration → Configuration groups*** предназначена для настройки конфиг-групп кластера.

Ansible settings

Вкладка ***Configuration → Ansible settings*** предназначена для ввода конфигурационных настроек Ansible на уровне кластера. Вкладка доступна начиная с версии ADCM 2.2.0.

Параметры Ansible

Имя	Описание	По умолчанию
forks	Число параллельных процессов, генерируемых при коммуникации с удаленными хостами	5

Импорт настроек ET

Ольга Семме, Дарья Барышева

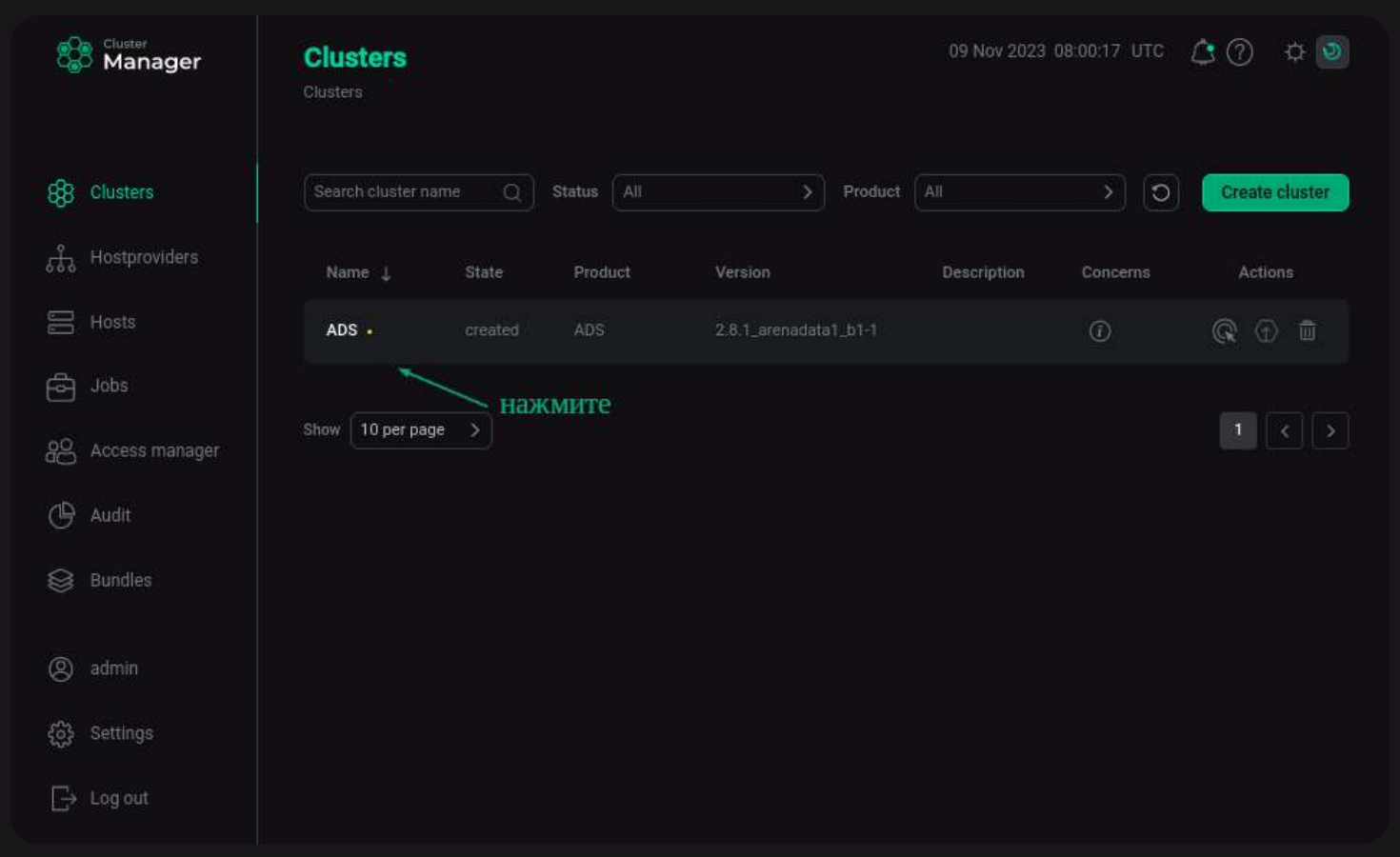
Назад к содержанию

ВАЖНО

Этот шаг необходим только в случае offline-установки.

Для установки кластера в режиме offline требуется интегрировать его с ранее созданным кластером Enterprise Tools (ET):

1. Выберите кластер на странице ***Clusters***. Для этого нажмите на имя кластера в столбце ***Name***.

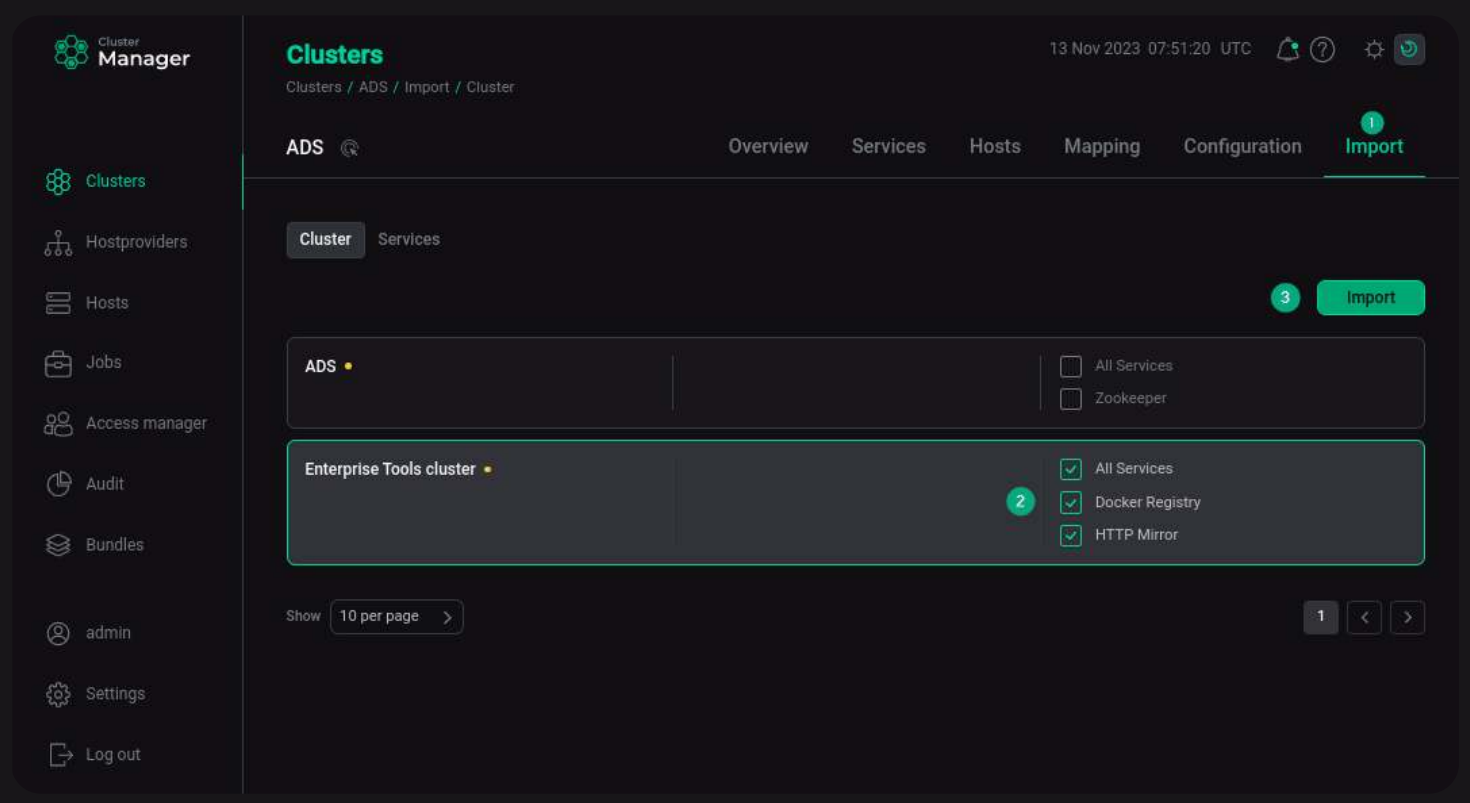


Выбор кластера

2. Откройте вкладку ***Import*** на странице кластера.

3. Выберите сервисы ***Docker Registry*** и ***HTTP Mirror*** предварительно созданного кластера ET.

4. Нажмите ***Import***.



Интеграция с кластером ET

ПРИМЕЧАНИЕ

После успешного импорта данные, хранимые локально в кластере Enterprise Tools, могут быть использованы на этапе установки продуктового кластера. Таким образом, их загрузка через Интернет не требуется.

Установка кластера

Д. Ольга Семин, Дарья Барышева

Содержание

Назад к содержанию

Шаг 1. Запуск установки
- Установка всего кластера
- Установка отдельных сервисов

Шаг 2. Просмотр результатов установки

Шаг 3. Проверка статуса кластера после установки

Шаг 1. Запуск установки

Существует два способа установки:

1. **Установка всего кластера**. В этом методе все сервисы кластера устанавливаются автоматически друг за другом после применения действия **Install** к кластеру.
2. **Установка отдельных сервисов**. В этом методе каждый сервис устанавливается вручную путем применения к нему одного или нескольких действий. Этот способ также можно использовать для добавления новых сервисов в уже развернутый кластер.

В обоим методам каждый сервис запускается автоматически после установки.

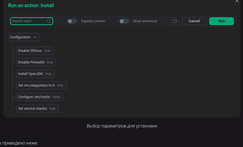
ВАЖНО

Установка должна производиться на чистые хосты без предустановленного подобного ПО.

Установка всего кластера

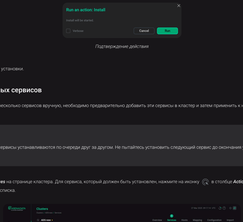
Чтобы установить все сервисы кластера за одну операцию, выполните шаги:

1. Примените действие **Install** к кластеру, выбрав/кнопку на странице **Clusters**, нажав на иконку ▶ в столбце **Actions**



Переход к установке всего кластера

2. В открывшемся диалоговом окне выберите необходимые параметры для установки и нажмите **Run**.

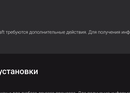


Выбор параметров для установки

Назначение параметров приведено ниже.

Параметр	Описание
Disable SELinux	Отключение SELinux на добавляемых хостах. Для того чтобы данная настройка применилась, после завершения операции **Install** необходимо перезагрузить хосты вручную
Disable Firewalld	Выключение firewalld на хостах
Install OpenJDK	Установка пакета java-1.8.0-openjdk на хостах
Set vm.swappiness to 0	Отключение аварийку на хостах
Configure /etc/hosts	Запись добавляемых нод в /etc/hosts на хостах кластера. Данную опцию рекомендуется отключить, если настроен DNS
Set service checks	Включит проверку сервиса при запуске действия

3. Подтвердите действие в открывшемся окне:



Подтверждение действия

4. Дождитесь завершения установки

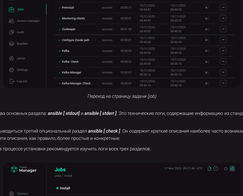
Установка отдельных сервисов

Чтобы установить один или несколько сервисов вручную, необходимо предварительно добавить эти сервисы в кластер и затем применить к ним действия, описанные ниже.

ВНИМАНИЕ

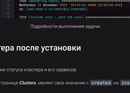
Обратите внимание, что сервисы устанавливаются по очереди друг за другом. Не пытайтесь установить следующий сервис до окончания установки предыдущего

1. Откройте вкладку **Services** на странице кластера. Для сервиса, который должен быть установлен, нажмите на иконку ▶ в столбце **Actions** и выберите действие **Install** из выпадающего списка.



Переход к установке отдельных сервисов

2. В открывшемся диалоговом окне выберите необходимые параметры для установки и нажмите **Run**.
3. Подтвердите действие в открывшемся окне:



Подтверждение действия

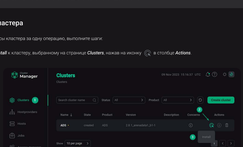
4. Дождитесь завершения установки и убедитесь, что статус сервиса изменился с `created` на значение `installed`.
5. Повторите предыдущие шаги для всех сервисов, которые необходимо установить.

ВАЖНО

- *При установке сервисов вручную важно соблюдать следующий порядок:*
  - ZooKeeper;
  - Kafka, NiFi (по одному);
  - другие сервисы.
- Во время установки сервиса Kafka в режиме KRaft требуются дополнительные действия. Для получения информации обратитесь к статье Установка Kafka в режиме KRaft.

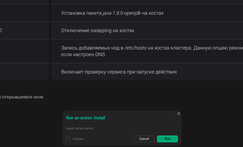
Шаг 2. Просмотр результатов установки

ADCM запускает отдельную задачу (job) для установки, как и для любого другого процесса. Для получения информации о статусе задач предназначена страница **Jobs**



Страница Jobs

Если установка завершается успешно, соответствующая ей задача на странице **Jobs** переходит из статуса `running` в `success`, в противном случае – в статус `failed`. Чтобы получить более детальную информацию о возможных ошибках, можно перейти на страницу задачи, кликнув по ней в списке задач на странице **Jobs**.

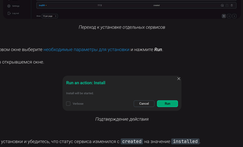


Переход на страницу задачи (job)

Страница задачи содержит два основных раздела: **ansible [stdout]** и **ansible [stderr]**. Это технические логи, содержащие информацию из стандартных I/O потоков ansible/ansible-playbook.

В дополнение к ним может выводиться третий специальный раздел **ansible [check]**. Он содержит краткое описание наиболее часто возникающих ошибок. По сравнению с предыдущими разделами, эти описания, как правило, более простые и конкретные.

При возникновении ошибок в процессе установки рекомендуется изучить логи всех трех разделов.

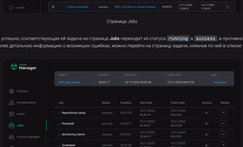


Подробности выполнения задачи

Шаг 3. Проверка статуса кластера после установки

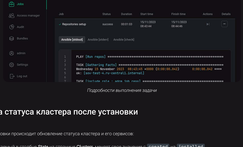
В результате успешной установки происходит обновление статуса кластера и его сервисов:

- Статус кластера, отображаемый в столбце **State** на странице **Clusters**, меняет свое значение с `created` на значение `installed`



Статус кластера после успешной установки

- Статус сервисов, отображаемый на вкладке **Services** страницы кластера, меняет свое значение с `created` на значение `installed`.



Статус сервисов кластера после успешной установки

Установка мониторинга

Ольга Семме

Назад к содержанию

Мониторинг — опциональный, но рекомендуемый компонент ADS, который можно настроить, используя один из следующих подходов:

- добавить в кластер ADS сервис Monitoring (доступен начиная с версии 3.6.2.2.b1);
- добавить сервисы мониторинга в кластер Enterprise Tools и интегрировать его с кластером ADS (поддержка этого способа установки мониторинга будет прекращена в будущих релизах ADS).

Сервис мониторинга

Артём Семенов

Содержание

Обзор
Шаг 1. Добавление сервиса
Шаг 2. Добавление компонентов
Шаг 3. Настройка сервиса
Шаг 4. Установка сервиса
Шаг 5. Проверка результатов
Данные в формате Prometheus
Веб-интерфейс Prometheus
Веб-интерфейс Grafana

Назад к содержанию

Обзор

При добавлении сервиса Monitoring в кластер K0S вся часть установляется на control plane. Этот компонент устанавливает с control plane системные метрики, которые будет собирать Prometheus. Prometheus, как и Grafana, также установлен на control plane. Prometheus собирает метрики с control plane, worker nodes, API, kubelet, Konnectivity, Calico, CoreDNS. Для настройки Grafana и формата Prometheus см. пункты с похожими темами (информер, развёртывание и конфигурирование Prometheus сервера).

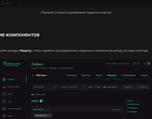
Для работы Prometheus, Grafana, которые собирают данные с кластера, нужно в веб-интерфейсе Prometheus или Grafana:

ПРИМЕЧАНИЕ

- Если узел, где установлен Prometheus, недоступен, метрики с кластера не будут собираться. Для этого нужно настроить [Prometheus federation](https://prometheus.io/docs/prometheus/latest/federation/) — механизм, позволяющий одному Prometheus-серверу собирать данные с других Prometheus-серверов. Это позволяет создавать иерархии и объединять данные с нескольких кластеров K0S.
- Вы также можете использовать [Thanos](https://thanos.io/), чтобы перенести все метрики из сервиса Prometheus, развёрнутого в K0S, в свой Prometheus.

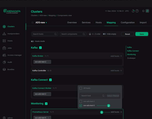
Шаг 1. Добавление сервиса

1. В интерфейсе K0S control plane в сервисе **Clusters** в элементе по имени кластера K0S. Затем во вкладке **Services** нажмите кнопку **Add service**.



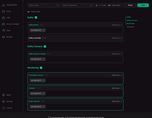
Переход к добавлению сервиса

2. В открывшемся окне выберите сервис **Monitoring** и нажмите **Add**.



Выбор сервиса

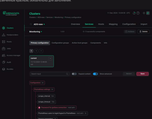
В результате добавленный сервис отобразится во вкладке **Services**.



Результат добавления сервиса в кластер

Шаг 2. Добавление компонентов

1. На странице настроек сервиса во вкладке **Mapping** чтобы перейти к расширенному справочнику компонентов между настройками сервиса.

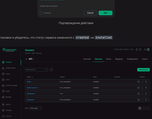


Переход к расширенному справочнику компонентов

Компоненты сервиса Monitoring

Компонент	Описание
Prometheus Server	Развёртывает в K0S сервер Prometheus, который выполняет следующие функции: • Агрегирование для всех кластеров, собирающих метрики в кластер. • хранилище всех метрик кластера. • Интерфейс для запросов на основе собранных метрик.
Grafana	Позволяет визуализировать метрики K0S в виде графиков и диаграмм, организованных в дашборды.
Node exporter	Экспортирует в Prometheus метрики узлов K0S.

2. Нажмите кнопку напротив компонента сервиса Monitoring — элемента **Add hosts** и выберите нужный хост из открывшегося окна.



Выбор хоста для компонента

ПРИМЕЧАНИЕ

Компоненты Prometheus Server не рекомендуется устанавливать на хост с K0S — это может работать для теста, но не для продакшена. Prometheus Server потребляет много ресурсов CPU и RAM, что может привести к нестабильной работе K0S. Рекомендуется выделять для Prometheus Server отдельные worker-машины.

3. После завершения распределения компонентов нажмите **Save**, чтобы сохранить изменения.

Сохранение распределения компонентов

Шаг 3. Настройка сервиса

1. Откройте вкладку **Services** на странице кластера и нажмите на имя сервиса **Monitoring** в столбце **Name**.

Переход к настройке сервиса

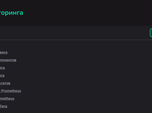
2. В открывшемся окне настроек сервиса Monitoring — конечном параметре конфигурации сервиса, в разделе **Monitoring** опции [конфигурационного справочника K0S](). Поля, перечисленные ниллсами, обязательны для заполнения.

Настройка сервиса Monitoring

После указания всех необходимых параметров нажмите **Save**.

Шаг 4. Установка сервиса

1. На вкладке **Services** в списке сервисов Monitoring появился по иконке **Actions** в контекстном меню выберите пункт **Install**.



Переход к установке сервиса

2. Подтвердите действие в открывшемся окне.

Подтверждение действия

3. Дождитесь завершения установки и убедитесь, что статус сервиса изменился на **installed** — **Uninstalled**.

Процесс установки завершён

Чтобы проконтролировать процесс выполнения установки сервиса на проблемы/ошибки, ошибки в случае их возникновения, выберите пункт **Jobs** в левом навигационном меню и нажмите на ссылку **Install** в поле задач **Jobs**.

Страница задачи Install

Шаг 5. Проверка результатов

Сервис Monitoring предоставляет метрики кластера и его элементов. По завершении установки необходимо убедиться в следующем:

- Метрики собираются по всем хостам кластера. Для этого в веб-интерфейсе Prometheus перейдите в раздел **Status** — **Targets** и убедитесь, что все цели имеют статус **UP**.
- С кластера корректно поступают системные метрики, а также метрики сервиса K0S — см. [Запрос системных метрик кластера K0S]().

Чтобы проверить, что данные с кластера поступают в веб-интерфейс Prometheus, в меню интерфейса веб-интерфейса Prometheus и Grafana.

ПРИМЕЧАНИЕ

Для работы мониторинга K0S на странице [Configuration](#) сервиса Kuffy перейдите по разделу **API Monitoring** и активируйте компоненты. После этого перезагрузите сервис.

Данные в формате Prometheus

В формате Prometheus можно получить метрики K0S, а также сервисов Kuberntes, развёрнутых в кластере. Для доступа к метрикам, которые предоставляет Prometheus, используется **Prometheus settings** — предоставляемый для настройки сервиса Prometheus **Kuyr-exporter settings**, предназначенный для настройки сервиса Prometheus.

Prometheus-сервер для Kuberntes-кластера K0S работает на [10.10.10.247](). Чтобы получить данные Prometheus:

1. Перейдите в веб-интерфейс Prometheus и нажмите на ссылку **Prometheus** в разделе **Prometheus**.

Переход к разделу Prometheus

2. В открывшемся окне выберите раздел **Prometheus settings** и нажмите на ссылку **Prometheus**.

Переход к разделу Prometheus

3. В поле **Expression** можно ввести название метрики, например, **Cluster** — в интерфейсе будут показаны значения этой метрики на всех кластерах K0S.

Веб-интерфейс Prometheus

Веб-интерфейс Prometheus

В веб-интерфейсе Prometheus можно получить метрики K0S, а также сервисов Kuberntes, развёрнутых в кластере. Для доступа к метрикам, которые предоставляет Prometheus, используется **Prometheus settings** — предоставляемый для настройки сервиса Prometheus **Kuyr-exporter settings**, предназначенный для настройки сервиса Prometheus.

Prometheus-сервер для Kuberntes-кластера K0S работает на [10.10.10.247](). Чтобы получить данные Prometheus:

1. Перейдите в веб-интерфейс Prometheus и нажмите на ссылку **Prometheus** в разделе **Prometheus**.

Переход к разделу Prometheus

2. В открывшемся окне выберите раздел **Prometheus settings** и нажмите на ссылку **Prometheus**.

Переход к разделу Prometheus

3. В поле **Expression** можно ввести название метрики, например, **Cluster** — в интерфейсе будут показаны значения этой метрики на всех кластерах K0S.

Веб-интерфейс Prometheus

Веб-интерфейс Grafana

В веб-интерфейсе Grafana можно получить метрики K0S, а также сервисов Kuberntes, развёрнутых в кластере. Для доступа к метрикам, которые предоставляет Grafana, используется **Grafana settings** — предоставляемый для настройки сервиса Grafana **Kuyr-exporter settings**, предназначенный для настройки сервиса Grafana. Grafana-сервер для Kuberntes-кластера K0S работает на [10.10.10.247:3000](). Для доступа к метрикам, которые предоставляет Grafana, используется **Grafana settings** — предоставляемый для настройки сервиса Grafana **Kuyr-exporter settings**, предназначенный для настройки сервиса Grafana.

1. В открывшемся окне выберите раздел меню **Home** — **Dashboards** и разверните раздел **Dashboard «K0S_cluster_name»**. В открывшемся окне выберите один из доступных дашбордов для просмотра метрик кластера или сервисов или нажмите на ссылку **New** для создания нового дашборда.

Переход к интерфейсу кластера K0S в Grafana

Установка мониторинга

Дарья Барышева

Назад к содержанию

Мониторинг является опциональным компонентом (однако, рекомендуемым). ADCM поддерживает настройку мониторинга на базе трех сервисов: Diamond, Graphite и Grafana.

Для процесса **offline**-установки необходимости в отдельном кластере мониторинга нет, так как его функции выполняет кластер Enterprise Tools (ET). Шаги для установки мониторинга в этом случае приведены ниже. Их можно выполнить до и после установки кластера ADS, кроме последнего шага Интеграция с кластером ADS, который может быть выполнен только на базе добавленного кластера ADS.

- Настройка кластера Enterprise Tools
- Интеграция с кластером ADS

ВНИМАНИЕ

Начиная с версии 3.6.2.2.b1 не рекомендуется настраивать мониторинг ADS через отдельный кластер мониторинга, так как с этого момента поддержка данной функциональности приостановлена и будет полностью прекращена в будущих релизах ADS. Рекомендуемый способ мониторинга кластера ADS — использовать сервис Monitoring.

Настройка кластера Enterprise Tools

© Арсен Сайдаметов

Содержание

Шаг 1. Добавление сервисов мониторинга в кластер ET
Шаг 2. Распределение компонентов мониторинга
Шаг 3. Настройка сервисов мониторинга
Шаг 4. Установка сервисов мониторинга
Шаг 5. Проверка результатов
- Graphite
- Grafana

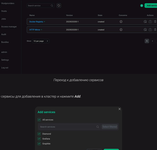
ВАЖНО!

Перед настройкой мониторинга убедитесь, что кластер Enterprise Tools (ET) успешно развёрнут в ADCM. Все последующие действия выполняются в нем.

Шаг 1. Добавление сервисов мониторинга в кластер ET

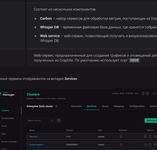
Для добавления сервисов в кластер:

1. Выберите кластер на странице **Clusters**. Для этого нажмите на имя кластера в столбце **Name**



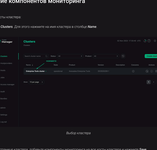
Выбор кластера

2. Откройте вкладку **Services** на странице кластера и нажмите **Add service**.



Переход к добавлению сервисов

3. В открывшемся окне выберите сервисы для добавления в кластер и нажмите **Add**



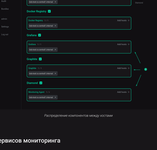
Выбор сервисов

Краткое описание доступных сервисов приведено ниже.

Сервисы мониторинга, которые необходимо добавить в кластер Enterprise Tools

Сервис	Назначение
Diamond	Python-сервис, используемый для сбора системных метрик с хостов, подключенных к системе мониторинга. Собранные метрики обрабатываются в Carbon (backend-сервисе Graphite).
Graphite	Состоит из нескольких компонентов: • **Carbon** — набор сервисов для обработки метрик, поступающих от Diamond; • **Whisper DB** — временная фиксация баз данных, где хранятся собранные метрики; • **Web service** — веб-сервис, позволяющий получать и визуализировать метрики из базы данных Whisper DB
Grafana	Web-сервис, предназначенный для создания графиков и оповещений для визуализации метрик, полученных из Graphite. По умолчанию используется порт **3000**

4. В результате успешно добавленные сервисы отобразятся на вкладке **Services**

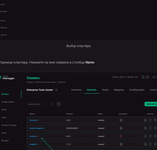


Результат успешного добавления сервисов в кластер

Шаг 2. Распределение компонентов мониторинга

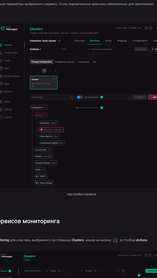
Для добавления компонентов на хосты кластера:

1. Выберите кластер на странице **Clusters**. Для этого нажмите на имя кластера в столбце **Name**



Выбор кластера

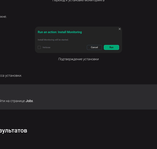
2. Откройте вкладку **Mapping** на странице кластера, добавьте компоненты на все хосты кластера и нажмите **Save**



Распределение компонентов между хостами

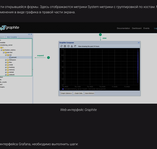
Шаг 3. Настройка сервисов мониторинга

1. Выберите кластер на странице **Clusters**. Для этого нажмите на имя кластера в столбце **Name**



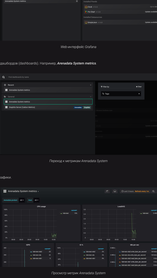
Выбор кластера

2. Откройте вкладку **Services** на странице кластера. Нажмите на имя сервиса в столбце **Name**



Переход к настройке сервисов

3. В открывшемся окне:
   - Откройте вкладку **Primary Configuration**
   - Переведите в активное состояние переключатель **Show advanced**
   - Заполните конфигурационные параметры (выделяются красным, обязательны для заполнения)
   - Нажмите **Save**



Настройка сервиса

Шаг 4. Установка сервисов мониторинга

1. Запустите действие **Install Monitoring** для кластера, выбрав на его странице **Clusters**, сначала на иконку ⚙, а потом **Actions**.

Переход к установке мониторинга

2. Нажмите **Run** в открывшемся окне.

Подтверждение установки

3. Дождитесь завершения процесса установки.

ПРИМЕЧАНИЕ

В случае ошибки logs можно найти на странице **Jobs**

Шаг 5. Проверка результатов

Graphite

Чтобы проверить доступность web-интерфейса Graphite, необходимо выполнить шаги:

1. В адресной строке браузера введите адрес хоста, на котором размещен кластер мониторинга. В качестве номера порта укажите значение параметра **Web interface TCP port** из секции **ui_ext_ports**, определённой на этапе конфигурирования сервиса Graphite при установке кластера мониторинга. Значение **80** использовалось по умолчанию (можно не указывать).

ВАЖНО!

В случае иных обстоятельств, уточните у администратора, какой из портов размещен на хосте.

2. Разверните дерево в левой части открывшейся формы. Здесь отображается метрическое дерево, структурированное по хостам. Можно выбрать любую из метрик для построения временного ряда и анализа в виде графика в правой части экрана.

Web-интерфейс Graphite

Grafana

Чтобы проверить доступность web-интерфейса Grafana, необходимо выполнить шаги:

1. В адресной строке браузера введите адрес хоста, на котором размещен кластер мониторинга. В качестве номера порта укажите значение параметра **Port** из раздела **ui_ext_ports**, определённого в конфигурации сервиса Grafana при установке кластера мониторинга (значение **3000** использовалось по умолчанию).
2. Выберите пункт меню **Home** в открывшемся окне.

Web-интерфейс Grafana

3. Откройте любой из доступных дашбордов (Dashboards), например, **Annushka System metrics**.

Переход к метрикам Annushka System

4. Проанализируйте доступные графики.

Просмотр метрик Annushka System

Интеграция с кластером ADS

Alexey Timanovsky, Digital Assistant

ADS

Содержание

Назад к содержанию

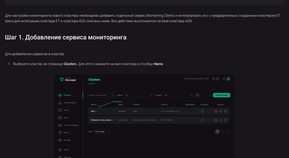
- Шаг 1. Добавление сервиса мониторинга
- Шаг 2. Распределение компонентов мониторинга
- Шаг 3. Импорт настроек мониторинга
- Шаг 4. Установка сервиса мониторинга
- Шаг 5. Проверка результатов
  - Graphite
  - Grafana

Для настройки мониторинга нового кластера необходимо добавить отдельный сервис Monitoring Clients и интегрировать его с предварительно созданным кластером ET. Шаги для интеграции кластера ET и кластера ADS описаны ниже. Все действия выполняются на базе кластера ADS.

Шаг 1. Добавление сервиса мониторинга

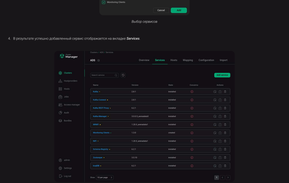
Для добавления сервиса в кластер:

1. Выберите кластер на странице **Clusters**. Для этого нажмите на имя кластера в столбце **Name**



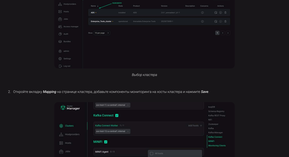
Выбор кластера

2. Откройте вкладку **Services** на странице кластера и нажмите **Add service**



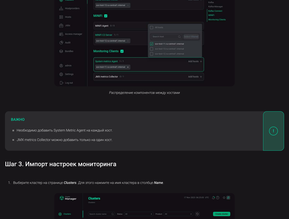
Переход к добавлению сервисов

3. Выберите сервис Monitoring Clients в открывшемся окне и нажмите **Add**



Выбор сервисов

4. В результате успешно добавленный сервис отобразится на вкладке **Services**

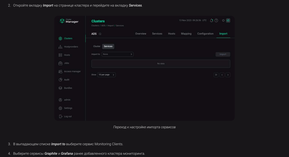


Результат успешного добавления сервиса в кластер

Шаг 2. Распределение компонентов мониторинга

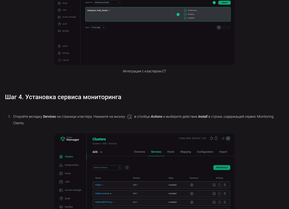
Для добавления компонентов на хосты кластера:

1. Выберите кластер на странице **Clusters**. Для этого нажмите на имя кластера в столбце **Name**



Выбор кластера

2. Откройте вкладку **Mapping** на странице кластера, добавьте компоненты мониторинга на хосты кластера и нажмите **Save**



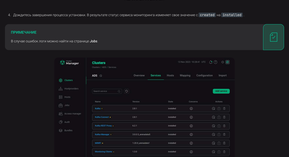
Распределение компонентов между хостами

ВАЖНО!

- Не следует добавлять System Metric Agent на каждый хост.
- JMX metrics Collector можно добавлять только на один хост.

Шаг 3. Импорт настроек мониторинга

1. Выберите кластер на странице **Clusters**. Для этого нажмите на имя кластера в столбце **Name**

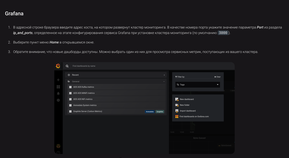


Выбор кластера

2. Откройте вкладку **Import** на странице кластера и перейдите на вкладку **Services**

Переход к настройке импорта сервисов

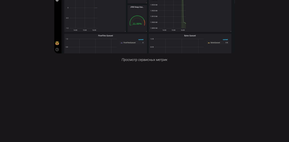
3. В выпадающем списке **Import** выберите сервис Monitoring Clients
4. Выберите сервисы **Graphite** и **Grafana** рядом добавленного мониторинга
5. Нажмите **Import**



Интеграция с кластером ET

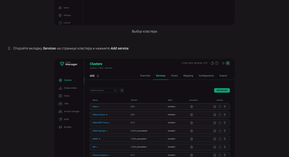
Шаг 4. Установка сервиса мониторинга

1. Откройте вкладку **Services** на странице кластера. Нажмите на иконку ⚙ в столбце **Actions** и выберите действие **Install** в группе, содержащей сервис Monitoring Clients



Переход к установке сервиса мониторинга

2. В открывшемся диалоговом окне выберите [автоподстроив параметры для установки](#) и нажмите **Run**
3. Подтвердите действие в открывшемся окне



Подтверждение действия

4. Дождитесь завершения процесса установки. В результате статус сервиса мониторинга изменит свое значение `STARTED` → `INSTALLED`

ПРИМЕЧАНИЕ

В случае ошибок, логи можно найти на странице **Jobs**

Процесс установки завершен

Шаг 5. Проверка результатов

По завершении установки необходимо убедиться в следующем:

- Системные метрики собираются с хостов вашего кластера, а не только с хоста, на котором расположен Ambari (проверяется настройками мониторинга).
- Наряду с системными метриками с хостов кластера поступают сервисные метрики. Например, кластер ADS обеспечивает метрики для таких сервисов как Kafka, AIS, Druid.

Оба пункта можно проверить в web-интерфейсе Graphite и Grafana.

Graphite

1. В адресной строке браузера введите адрес хоста, на котором развернут кластер мониторинга. В качестве номера порта укажите значение параметра **Web-interface TCP port** из раздела **gc_and_ports**, определенное на этапе конфигурирования сервиса Graphite при установке кластера мониторинга. Значение **2003** зарезервировано по умолчанию) нажмите на **Enter**.
2. Разверните дерево в левой части открывшейся формы. Здесь отображаются метрики System и сервисные метрики с группировкой по хостам. Можно выбрать любые из метрик для построения динамики их изменения в виде графика в правой части формы.

Новые хосты и сервисные метрики доступны в Graphite

Grafana

1. В адресной строке браузера введите адрес хоста, на котором развернут кластер мониторинга. В качестве номера порта укажите значение параметра **Port** из раздела **gc_and_ports**, определенное на этапе конфигурирования сервиса Grafana при установке кластера мониторинга (по умолчанию - **3000**)
2. Выберите пункт меню **Home** и открывшийся dash.
3. Обратите внимание, что новые дашборды доступны. Можно выбрать один из них для просмотра сервисных метрик, поступающих со вашего кластера.

Переход к сервисным метрикам

4. Проанализируйте доступные графики

Просмотр сервисных метрик

Начало работы с Kafka

Ольга Синяк

Содержание

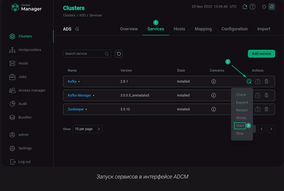
Назад к содержанию

- Установка и запуск среды Kafka с использованием ADCM
- Подключение к Kafka
- Создание топика
- Запись сообщения в топик
- Чтение сообщений из топика
  - Чтение сообщений с начала партиции
  - Чтение сообщений с заданного смещения
- Завершение работы в среде Kafka

В данной статье описаны первые шаги для начала работы с Kafka.

Установка и запуск среды Kafka с использованием ADCM

1. Для установки и запуска среды Kafka с использованием ADCM в кластере ADS необходимо установить сервисы:
   - Kafka
   - ZooKeeper
2. После установки в интерфейсе ADCM для всех установленных сервисов необходимо по очереди применить действие **Start**, нажав на иконку в столбце **Actions** и дождаться успешного завершения запуска.



Запуск сервисов в интерфейсе ADCM

В результате на соответствующих хостах установлена среда Kafka и запускаются серверы Kafka broker и ZooKeeper.

ПРИМЕЧАНИЕ

Установка сервисов описана в статьях Online-установка, Offline-установка.

Подключение к Kafka

При установке Kafka на хостах создается новая директория /usr/lib/kafka/bin. Она содержит файлы с расширением .sh – скрипты для выполнения команд в среде Kafka. Для просмотра директории необходимо выполнить следующую команду:

```
$ ls -la /usr/lib/kafka/bin
```

Работа с сервисом Kafka происходит путем запуска соответствующих скриптов в командной строке. Обычно скрипты запускают, находясь в папке */usr/lib/kafka*, указывая в команде часть пути *bin/*. Перейти в нужную директорию можно, используя команду:

```
$ cd /usr/lib/kafka
```

Создание топика

Для того чтобы записать сообщения в Kafka, необходимо создать топик. Это осуществляется при помощи скрипта *kafka-topics.sh* и опции `--create` с любого хоста кластера, на котором установлен сервис Kafka.

Для создания топика необходимо выполнить следующую команду:

```
$ bin/kafka-topics.sh --create --topic <topic_name> --bootstrap-server <hostname>:9092
```

где (в данной команде и ниже):

- `<topic_name>` – название топика;
- `<hostname>` – имя хоста, где создается топик;
- `9092` – HTTP-порт доступа к сервису Kafka.

В результате появляется сообщение:

```
Created topic <topic_name>.
```

Для просмотра данных о существующем топике используется скрипт *kafka-topics.sh* и опция `--describe`.

Для просмотра данных о топике необходимо выполнить следующую команду:

```
$ bin/kafka-topics.sh --describe --topic <topic_name> --bootstrap-server <hostname>:9092
```

В результате выводится данные о запрошенном топике в виде:

```
Topic: <topic_name>     TopicId: STGaaB0QSFK4s0dMRtp-YA PartitionCount: 1       ReplicationFactor: 1    Configs: unclean.leader.
election.enable=false
Topic: <topic_name>     Partition: 0    Leader: 1001    Replicas: 1001 Isr: 1001
```

Здесь указано, что созданный топик обладает следующими параметрами:

- количество партиций – `1`;
- коэффициент репликации – `1`;
- обозначение брокера-лидера для данной партиции – `1001`;
- настройка `unclean.leader.election.enable` выключена.

ПРИМЕЧАНИЕ

Для просмотра других действий, которые выполняет скрипт *kafka-topics.sh*, необходимо выполнить команду:

```
$ bin/kafka-topics.sh
```

Запись сообщения в топик

Запись сообщения в топик производится при помощи скрипта *kafka-console-producer.sh* с любого хоста кластера, на котором установлен сервис Kafka.

Для записи сообщения в топик `<topic_name>` необходимо:

1. Выполнить следующую команду:

```
$ bin/kafka-console-producer.sh --topic <topic_name> --bootstrap-server <hostname>:9092
```

   При этом запускается режим записи сообщений.

2. На следующей строке после ввода команды ввести необходимые сообщения, каждое на новой строке. Запись сообщения и переход на новую строку производятся при помощи `Enter`. Например, таким образом вводятся несколько сообщений, каждое на своей строке:

```
>Sunday
>Monday
>Tuesday
>Wednesday
>Thursday
>Friday
>Saturday
```

3. Выйти из режима записи сообщений, для этого необходимо:
   - после записи последнего сообщения перейти на следующую строку;
   - нажать `Ctrl+C`.

ПРИМЕЧАНИЕ

Для просмотра других опций, которые можно применить при выполнении скрипта *kafka-console-producer.sh*, необходимо выполнить команду:

```
$ bin/kafka-console-producer.sh
```

Чтение сообщений из топика

Чтение сообщений из топика производится при помощи скрипта *kafka-console-consumer.sh* с любого хоста кластера, на котором установлен сервис Kafka.

Чтение сообщений с начала партиции

Для чтения сообщений из топика `<topic_name>` с самого первого сообщения, записанного в партицию, необходимо выполнить следующую команду с использованием опции `--from-beginning`:

```
$ bin/kafka-console-consumer.sh --topic <topic_name> --from-beginning --bootstrap-server <hostname>:9092
```

При этом запускается режим чтения сообщений. Сообщения считываются в том же порядке, как они были записаны:

```
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
```

Выход из режима чтения сообщений осуществляется при помощи `Ctrl+C`.

РЕКОМЕНДАЦИЯ

Если в этот же топик при помощи *kafka-console-producer.sh* с другой терминальной сессии записывать сообщения, в режиме чтения можно их сразу считывать.

Чтение сообщений с заданного смещения

Для чтения сообщений из топика `<topic_name>`, начиная со смещения 4 (`Thursday`), необходимо выполнить следующую команду с использованием опций указания партиции `--partition 0` и смещения `--offset 4`:

```
$ bin/kafka-console-consumer.sh --topic <topic_name> --partition 0 --offset 4 --bootstrap-server <hostname>:9092
```

Результатом будет вывод сообщений, записанных начиная со смещения 4:

```
Thursday
Friday
Saturday
```

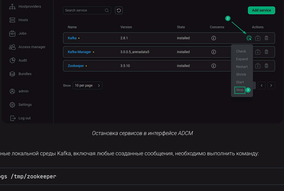
ПРИМЕЧАНИЕ

Для просмотра других опций, которые можно применить при выполнении скрипта *kafka-console-consumer.sh*, необходимо выполнить команду:

```
$ bin/kafka-console-consumer.sh
```

Завершение работы в среде Kafka

Для остановки сервисов в интерфейсе ADCM для всех сервисов по очереди необходимо применить действие **Stop**, нажав на иконку в столбце **Actions** и дождаться успешного завершения действия.



Остановка сервисов в интерфейсе ADCM

Если необходимо удалить данные локальной среды Kafka, включая любые созданные сообщения, необходимо выполнить команду:

```
$ rm -rf /tmp/kafka-logs /tmp/zookeeper
```

Использование SASL PLAINTEXT в Kafka

Ольга Семко

Содержание

Назад к содержанию

Проверка включения аутентификации SASL PLAINTEXT

Управление пользователями с аутентификацией SASL PLAINTEXT в Kafka

- Создание пользователя
- Удаление и изменение данных пользователя

Аутентификация пользователя в среде Kafka

Работа с .sh файлами (скриптами) с участием созданного пользователя

ПРИМЕЧАНИЕ

Включение аутентификации SASL PLAINTEXT осуществляется в соответствии со статьей SASL PLAINTEXT.

Проверка включения аутентификации SASL PLAINTEXT

1. Проверьте конфигурацию безопасности и аутентификации брокеров Kafka.
   На каждом хосте с брокером Kafka введите команду:

```
$ vim /etc/kafka/conf/server.properties
```

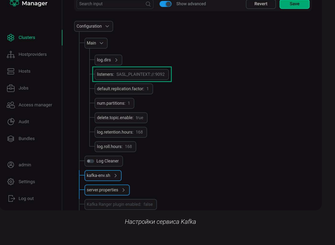
   Убедитесь, что в файле *server.properties* для каждого брокера Kafka строки, определяющие протокол безопасности, изменены на `SASL_PLAINTEXT`; строки, определяющие механизм аутентификации – на `PLAIN`, а также сгенерированы пароли для всех сервисов кластера ADS:

```
security.inter.broker.protocol=SASL_PLAINTEXT


sasl.mechanism.inter.broker.protocol=PLAIN
sasl.enabled.mechanisms=PLAIN

listener.name.sasl_plaintext.plain.sasl.jaas.config=org.apache.kafka.common$
  username=kafka \
  password="kafka-password" \
  user_kafka="kafka-password" \
  user_ksql-server="ksql-server-password" \
  user_kafka-rest="kafka-rest-password" \
  user_schema-registry="schema-registry-password" \
  user_kafka-connect="kafka-connect-password";
```

2. Проверьте, что после установки SASL PLAINTEXT в настройках сервиса Kafka изменился параметр *listeners* : `PLAINTEXT://:9092` на `SASL_PLAINTEXT://:9092`.



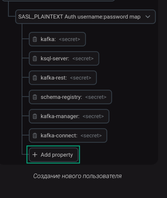
Настройки сервиса Kafka

Управление пользователями с аутентификацией SASL PLAINTEXT в Kafka

Создание пользователя

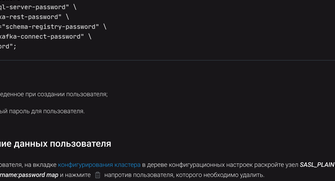
Для того чтобы создать пользователя:

1. На вкладке конфигурирования кластера в дереве конфигурационных настроек раскройте узел ***SASL_PLAINTEXT Authentication***, далее узел ***SASL_PLAINTEXT Auth username.password map*** и нажмите ***+ Add property*** для создания нового пользователя.



Создание нового пользователя

2. В открывшемся окне введите имя пользователя (field name), пароль (secret), подтвердите пароль и нажмите ***Apply***.
   В имени пользователя допускается использовать строчные буквы, цифры, точку, подчеркивание и/или дефис.



Имя и пароль для нового пользователя

3. После ввода данных сохраните конфигурацию, кликнув ***Save***.
4. Перезагрузите сервис Kafka. Для этого примените действие Restart, нажав на иконку в столбце ***Actions***.
5. Убедитесь, что в файле *server.properties* для каждого брокера Kafka появилась запись о новом пользователе:

```
listener.name.sasl_plaintext.plain.sasl.jaas.config=org.apache.kafka.common$
  username=kafka \
  password="kafka-password" \
  user_kafka="kafka-password" \
  user_ksql-server="ksql-server-password" \
  user_kafka-rest="kafka-rest-password" \
  user_schema-registry="schema-registry-password" \
  user_kafka-connect="kafka-connect-password" \
  user_new_user="password";
```

   где:

   - `new_user` – имя, введенное при создании пользователя;
   - `password` – заданный пароль для пользователя.

Удаление и изменение данных пользователя

Для того чтобы удалить пользователя, на вкладке конфигурирования кластера в дереве конфигурационных настроек раскройте узел ***SASL_PLAINTEXT Authentication***, далее узел ***SASL_PLAINTEXT Auth username.password map*** и нажмите напротив пользователя, которого необходимо удалить.

Для изменения данных пользователя выберите нужного пользователя, внесите новые данные и нажмите ***Apply***.

После удаления пользователя или изменения данных:

1. Сохраните конфигурацию, кликнув ***Save***.
2. Перезагрузите сервис Kafka. Для этого примените действие Restart, нажав на иконку в столбце ***Actions***.

Аутентификация пользователя в среде Kafka

Для аутентификации каждого пользователя, в том числе автоматически созданного системой пользователя `kafka`, требуется создать специальный файл конфигурации *client.properties*. Для этого выполните шаги:

1. Откройте файл-заглушку, выполнив команду:

```
$ sudo vim /etc/kafka/conf/client.properties
```

2. Внесите изменения в файл, раскомментировав строку с параметром `sasl.jaas.config` и внеся данные о пользователе и пароле пользователя, как указано ниже:

```
security.protocol=SASL_PLAINTEXT
sasl.mechanism=PLAIN
# Uncomment and set necessary username/password
sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required \
 username="new_user" \
 password="password";
```

   где имя пользователя и пароль должны быть указаны в кавычках, как указано в примере.

Работа с .sh файлами (скриптами) с участием созданного пользователя

1. Создайте топик, указав при помощи опции `--command-config` путь к файлу конфигурации пользователя *client.properties*, описанному выше:

```
$ /usr/lib/kafka/bin/kafka-topics.sh --create --topic test-topic --bootstrap-server localhost:9092 --command-config
/etc/kafka/conf/client.properties
```

   В случае успешного создания топика выводится сообщение:

```
Created topic test-topic.
```

2. Запишите сообщения в топик, указав при помощи опции `--producer.config` путь к файлу конфигурации пользователя *client.properties*, описанному выше:

```
$ /usr/lib/kafka/bin/kafka-console-producer.sh --topic test-topic --bootstrap-server localhost:9092 --producer.config
/etc/kafka/conf/client.properties
```

3. Считайте сообщения из топика, указав при помощи опции `--consumer.config` путь к файлу конфигурации пользователя *client.properties*, описанному выше:

```
$/usr/lib/kafka/bin/kafka-console-consumer.sh --topic test-topic --from-beginning  --bootstrap-server sov-ads-test-4.ru-central1.
internal:9092 --consumer.config /etc/kafka/conf/client.properties
```

   Убедитесь в корректности считанных сообщений.

Попытка запуска любого скрипта от имени пользователя, не внесенного в файл *server.properties*, вызовет ошибку аутентификации, например:

```
[2022-12-01 13:42:25,572] ERROR [AdminClient clientId=adminclient-1] Connection to node -1 (sov-ads-test-4.ru-central1.
internal/10.92.16.223:9092) failed authentication due to: Authentication failed: Invalid username or password (org.apache.kafka.
clients.NetworkClient)
[2022-12-01 13:42:25,574] WARN [AdminClient clientId=adminclient-1] Metadata update failed due to authentication error (org.
apache.kafka.clients.admin.internals.AdminMetadataManager)
org.apache.kafka.common.errors.SaslAuthenticationException: Authentication failed: Invalid username or password
Error while executing topic command : Authentication failed: Invalid username or password
[2022-12-01 13:42:25,579] ERROR org.apache.kafka.common.errors.SaslAuthenticationException: Authentication failed: Invalid
username or password
```

Использование MIT Kerberos в Kafka

Ольга Синяк

Содержание

Назад к содержанию

Проверка установленного Kerberos SASL
Настройка учетных записей для аутентификации в Kafka
Создание JAAS-файла для пользователя
Создание файла конфигурации .properties для пользователя
Подключение к Kafka (создание тикетов) и работа с .sh файлами (скриптами) с участием разных пользователей

В данной статье описаны первые шаги в Kafka с установленным Kerberos SASL.

ПРИМЕЧАНИЕ

Включение аутентификации про протоколу Kerberos SASL осуществляется в соответствии со статьей MIT Kerberos.

Проверка установленного Kerberos SASL

1. Проверьте настройки, установленные для Kerberos в конфигурации кластера.
   Зайдите в конфигурации кластера, выберите **Show advanced**, найдите раздел с конфигурациями Kerberos, разверните его.

Установленные настройки конфигурации кластера ADS для Kerberos

2. Проверьте конфигурацию безопасности и аутентификации брокеров Kafka.
   На каждом хосте с брокером Kafka введите команду:

```
$ sudo vim /usr/lib/kafka/config/server.properties
```

Убедитесь, что в файле *server.properties* для каждого брокера Kafka строки, определяющие протоколы безопасности, изменены на `SASL_PLAINTEXT` строки, определяющие механизм аутентификации - на `GSSAPI`:

```
security.inter.broker.protocol=SASL_PLAINTEXT


sasl.mechanism.inter.broker.protocol=GSSAPI
sasl.enabled.mechanisms=GSSAPI
```

3. Проверьте, что после установки Kerberos SASL в настройках сервиса Kafka в группе ***server.properties*** изменился параметр ***listeners*** : `PLAINTEXT://:9092` -на `SASL_PLAINTEXT://:9092`.
4. Проверьте наличие созданных принципалов для каждого сервиса.
   На хосте с установленным сервером KDC введите команду:

```
$ sudo kadmin.local list_principals
```

Вывод списка принципалов показывает, что созданы принципалы для каждого хоста каждого установленного сервиса, принципалы админов и другие технические принципалы для созданной области:

```
K/M@ADS-KAFKA.LOCAL
admin/adsk@ADS-KAFKA.LOCAL
kadmin/admin@ADS-KAFKA.LOCAL
kadmin/changepw@ADS-KAFKA.LOCAL
kadmin/sov-ads-test-0.ru-central1.internal@ADS-KAFKA.LOCAL
kafka/sov-ads1-test.ru-central1.internal@ADS-KAFKA.LOCAL
kafka/sov-ads2-test.ru-central1.internal@ADS-KAFKA.LOCAL
kafka/sov-ads3-test.ru-central1.internal@ADS-KAFKA.LOCAL
kiprop/sov-ads-test-0.ru-central1.internal@ADS-KAFKA.LOCAL
krbtgt/ADS-KAFKA.LOCAL@ADS-KAFKA.LOCAL
zookeeper/sov-ads1-test.ru-central1.internal@ADS-KAFKA.LOCAL
zookeeper/sov-ads2-test.ru-central1.internal@ADS-KAFKA.LOCAL
zookeeper/sov-ads3-test.ru-central1.internal@ADS-KAFKA.LOCA
```

5. *Проверьте наличие на хостах с установленными сервисами файлов для хранения паролей *.service.keytab*.
   На каждом хосте с установленными сервисами введите команду:

```
$ ls -la /etc/security/keytabs/
```

Вывод списка файлов показывает, что созданы файлы *.service.keytab для каждого сервиса, установленного на хост:

```
total 16
drwxr-xr-x. 2 root      root       102 Aug  9 20:55 .
drwxr-xr-x. 7 root      root       4096 Aug  9 19:30 ..
-rw-r-----. 1 kafka     kafka      826 Aug  9 20:56 kafka.service.keytab
-rw-------. 1 zookeeper  zookeeper  858 Aug  9 20:55 zookeeper.service.keytab
```

Настройка учетных записей для аутентификации в Kafka

Для производителя (producer) и потребителя (consumer) можно создать принципалы соответствующих пользователей.

Для этого поочередно введите команды, используя имя пользователя и пароль.

```
$ sudo kadmin.local -q "add_principal -pw PASSWORD reader@ADS-KAFKA.LOCAL"
$ sudo kadmin.local -q "add_principal -pw PASSWORD writer@ADS-KAFKA.LOCAL"
```

Получите для каждой созданной учетной записи соответствующий результат:

```
Principal "reader@ADS-KAFKA.LOCAL" created.
Principal "writer@ADS-KAFKA.LOCAL" created.
```

Создание JAAS-файла для пользователя

Для всех принципалов должен быть создан JAAS-файл (Java Authentication and Authorization Service). В нем указывается, как будут использованы тикеты для конкретного принципала.

ПРИМЕЧАНИЕ

- Описание JAAS-файла и применимых опций приведены в статье KAS-sysMoloko.
- Данные клиентских подключений можно также указать в разделе `KafkaClient` файла *kafka-jaas.conf* при использовании шаблона JAAS-файла для сервиса Kafka.

Для принципалов брокеров после кербиризации автоматически создаются параметры подключения в файле *kafka-jaas.conf*. Для того чтобы посмотреть содержимое файла, введите команду:

```
$ sudo vim /usr/lib/kafka/config/kafka-jaas.conf
```

Для новых принципалов `writer` и `reader` самостоятельно создайте JAAS-файл.

1. Выполните команду:

```
$ sudo vim /tmp/client.jaas
```

2. Введите данные в файл:

```
KafkaClient {
com.sun.security.auth.module.Krb5LoginModule required
useTicketCache=true;
};
```

где `useTicketCache` – параметр, определяющий, будет ли тикет для этого пользователя получен из кеша тикетов. При установке данного параметра в `true` перед подключением в Kafka необходимо создать тикет пользователя.

Создание файла конфигурации .properties для пользователя

Для создания файла конфигурации *.properties* для пользователя выполните команду:

```
$ sudo vim /tmp/client.properties
```

Заполните файл данными:

```
security.protocol=SASL_PLAINTEXT
sasl.mechanism=GSSAPI
sasl.kerberos.service.name=kafka
```

Подключение к Kafka (создание тикетов) и работа с .sh файлами (скриптами) с участием разных пользователей

1. Откройте терминальную сессию 1 и подключитесь к одному из брокеров Kafka.
2. Создайте тикет для пользователя `writer`:

```
$ kinit -p writer@ADS-KAFKA.LOCAL
```

Введите пароль, указанный при создании пользователя.

3. Проверьте тикет:

```
$ klist
```

```
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: writer@ADS-KAFKA.LOCAL

Valid starting     Expires            Service principal
08/10/2022 20:44:12  08/11/2022 20:44:12  krbtgt/ADS-KAFKA.LOCAL@ADS-KAFKA.LOCAL
```

ПРИМЕЧАНИЕ

Полное описание функций команды *kinit* и применимых опций приведены по ссылке kinit.

4. Экспортируйте созданный файл *client.jaas* как параметр JVM для данного пользователя при помощи переменной среды `KAFKA_OPTS`:

```
$ export  KAFKA_OPTS="-Djava.security.auth.login.config=/tmp/client.jaas"
```

5. Создайте топик, указав путь к созданному файлу *client.properties*:

```
$ /usr/lib/kafka/bin/kafka-topics.sh --create --topic test-topic --bootstrap-server sov-ads-test-1.ru-central1.internal:9092,sov-ads-test-2.ru-central1.internal:9092,sov-ads-test-3.ru-central1.internal:9092 --command-config /tmp/client.properties
```

Получите подтверждение:

```
Created topic test-topic.
```

6. Запишите сообщение в топик, указав путь к созданному файлу *client.properties*:

```
$ /usr/lib/kafka/bin/kafka-console-producer.sh --topic test-topic --bootstrap-server sov-ads-test-1.ru-central1.internal:9092,sov-ads-test-2.ru-central1.internal:9092,sov-ads-test-3.ru-central1.internal:9092 --producer.config /tmp/client.properties
```

```
>One
>Two
>Three
>Four
>Five
```

7. Откройте терминальную сессию 2 и подключитесь к одному из брокеров Kafka.
8. Создайте тикет для пользователя `reader`:

```
$ kinit -p reader@ADS-KAFKA.LOCAL
```

Введите пароль, указанный при создании пользователя.

9. Проверьте тикет:

```
$ klist
```

```
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: reader@ADS-KAFKA.LOCAL

Valid starting     Expires            Service principal
08/10/2022 21:30:47  08/11/2022 21:30:47  krbtgt/ADS-KAFKA.LOCAL@ADS-KAFKA.LOCAL
```

10. Экспортируйте созданный файл *client.jaas* как параметр JVM для данного пользователя при помощи переменной среды `KAFKA_OPTS`:

```
$ export  KAFKA_OPTS="-Djava.security.auth.login.config=/tmp/client.jaas"
```

11. Считайте сообщения из топика, указав путь к созданному файлу *client.properties*:

```
$/usr/lib/kafka/bin/kafka-console-consumer.sh --topic test-topic --from-beginning  --bootstrap-server sov-ads-test-1.ru-central1.internal:9092,sov-ads-test-2.ru-central1.internal:9092,sov-ads-test-3.ru-central1.internal:9092 --consumer.config /tmp/client.properties
```

Полученные сообщения:

```
One
Two
Three
Four
Five
```

Убедитесь в корректности принятых сообщений.

Использование Kerberos с MS Active Directory в Kafka

Д. Сотых Семен

Содержание

Назад к содержанию

Проверка установленного Kerberos

Подключение клиента к Kafka

Создание JAAS-файла для пользователя

Создание файла конфигурации .properties для пользователя

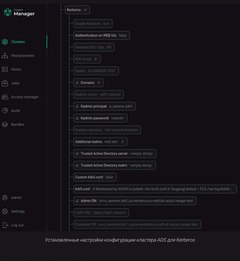
Подключение пользователя к Kafka (создание тикетов) и работа с .sh файлами (скриптами)

В данной статье описаны первые шаги в Kafka с настроенной Kerberos-аутентификацией на базе MS Active Directory.

> **ПРИМЕЧАНИЕ**
>
> - Включение аутентификации по протоколу Kerberos с MS Active Directory осуществляется в соответствии со статьей kerberos с MS Active Directory.
> - Описание, приведенное в этой статье, также соответствует работе в Kafka после включения аутентификации Kerberos с Samba.

Проверка установленного Kerberos

1. Проверьте настройки, установленные для Kerberos в конфигурациях кластера.
   Зайдите в конфигурации кластера, выберите ***Show advanced***, найдите раздел с конфигурациями Kerberos, разверните его.



Установленные настройки конфигурации кластера ADS для Kerberos

2. Проверьте конфигурацию безопасности и аутентификации брокеров Kafka.
   На каждом хосте с брокером Kafka введите команду:

```
$ sudo vim /usr/lib/kafka/config/server.properties
```

   Убедитесь, что в файле server.properties для каждого брокера Kafka строки, определяющие протокол безопасности, изменены на `SASL_PLAINTEXT`. строки, определяющая механизм аутентификации - на `GSSAPI`

```
security.inter.broker.protocol=SASL_PLAINTEXT


sasl.mechanism.inter.broker.protocol=GSSAPI
sasl.enabled.mechanisms=GSSAPI
```

3. Проверьте, что после установки Kerberos SASL в настройках сервиса Kafka в группе ***server.properties*** изменился параметр ***listeners*** : `PLAINTEXT://:9092` на `SASL_PLAINTEXT://:9092`.
4. При помощи утилиты командной строки для поиска в каталоге LDAP `ldapsearch` с хоста брокера Kafka существует возможность проверить созданные принципалы в среде `AD.RANGER-TEST` и владельца `Container DN`
   Вместо команды, указав `Admin DN` и `Container DN` и пароль пользователя, указанные при запуске LDAP на ADCM.

```
$ ldapsearch -v -H ldaps://ldap_host -x -D "cn=admin,ou=workforce,ou=adh,dc=ad,dc=ranger-test" -b "ou=admin,ou_ou=workforce,ou=adh,dc=ad,dc=ranger-test" -w Password
```

   В результате выведется информация по каждой созданной учетной записи в LDAP для каждого хоста каждого сервиса ADS. Вариант вывода для одной учетной записи:

```
# kafka/sov-ads-test-5.ru-central1.internal, admin_ou, kerberos, adh, ad.ranger-test
dn: CN=kafka/sov-ads-test-5.ru-central1.internal,OU=admin_ou,OU=kerberos,OU=adh,DC=ad,DC=ranger-test
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
cn: kafka/sov-ads-test-5.ru-central1.internal
distinguishedName: CN=kafka/sov-ads-test-5.ru-central1.internal,OU=admin_ou,OU=kerberos,OU=adh,DC=ad,DC=ranger-test
instanceType: 4
whenCreated: 20220816100038.0Z
whenChanged: 20220816100042.0Z
uSNCreated: 14958769
uSNChanged: 14958786
name: kafka/sov-ads-test-5.ru-central1.internal
objectGUID:: fJ7BUXMzUEaqUwqvKUO2MQ==
userAccountControl: 66048
badPwdCount: 0
codePage: 0
countryCode: 0
badPasswordTime: 0
lastLogoff: 0
lastLogon: 133853546143620502
pwdLastSet: 133851176304557312
primaryGroupID: 513
objectSid: AQUAAAAAAAUVAAAAAKiXX3eSBwdKCbr0VsQGAA==
accountExpires: 0
logonCount: 50
sAMAccountName: $IUV110-ESG0NALCFJGL
sAMAccountType: 805306368
userPrincipalName: kafka/sov-ads-test-5.ru-central1.internal@AD.RANGER-TEST
servicePrincipalName: kafka/sov-ads-test-5.ru-central1.internal
objectCategory: CN=Person,CN=Schema,CN=Configuration,DC=ad,DC=ranger-test
dSCorePropagationData: 16010101000000.0Z
lastLogonTimestamp: 133851174422039302
```

> **ПРИМЕЧАНИЕ**
>
> Полное описание функций утилиты командной строки `ldapsearch` и применимых опций приведены по ссылке ldapsearch. Синтаксис команды `ldapsearch` может отличаться в зависимости от используемой ОС.

5. Проверьте наличие на хостах с установленными сервисами файлов для хранения паролей * service.keytab
   На каждом хосте с установленными сервисами введите команду:

```
$ ls -la /etc/security/keytabs/
```

   Вывод списка файлов показывает, что созданы файлы *.service.keytab для каждого сервиса, установленного на хост:

```
total 16
drwxr-xr-x. 2 root      root      182 Aug  9 20:55
drwxr-xr-x. 7 root      root     4096 Aug  9 19:38 .
-rw-------. 1 kafka     kafka     826 Aug  9 20:54 kafka.service.keytab
-rw-------. 1 zookeeper zookeeper  858 Aug  9 20:55 zookeeper.service.keytab
```

Подключение клиента к Kafka

По умолчанию каждый пользователь из базы данных службы каталогов Active Directory, имеющий принципал в заданной области (realm), имеет права на подключение к кластеру Kafka и выполнение действий с топиками.

Создание JAAS-файла для пользователя

Для всех принципалов должен быть создан JAAS-файл (Java Authentication and Authorization Service). В нем указывается, как будут использованы тикеты для конкретного принципала.

> **ПРИМЕЧАНИЕ**
>
> - Описание JAAS-файла и применяемых опций приведены в статье KrbSLoginModule.
> - Данные клиентских подключений можно также указать в разделе `KafkaClient` файла kafka-jaas.conf при использовании шаблона JAAS-файла для сервиса Kafka.

Для принципалов брокеров после кербероизации автоматически создаются параметры подключения в файле kafka-jaas.conf. Для того чтобы посмотреть содержимое файла, введите команду:

```
$ sudo vim /usr/lib/kafka/config/kafka-jaas.conf
```

Для принципалов клиента необходимо самостоятельно создать файл JAAS.

1. Выполните команду:

```
$ sudo vim /tmp/client.jaas
```

2. Введите данные в файл:

```
KafkaClient {
com.sun.security.auth.module.Krb5LoginModule required
useTicketCache=true;
};
```

   где `useTicketCache` - параметр, определяющий будет ли тикет для этого пользователя получен из кэша тикетов. При установке данного параметра в `true` перед подключением в Kafka необходимо создать тикет пользователя.

Создание файла конфигурации .properties для пользователя

Для создания файла конфигурации .properties для пользователя выполните команду:

```
$ sudo vim /tmp/client.properties
```

Заполните файл данными:

```
security.protocol=SASL_PLAINTEXT
sasl.mechanism=GSSAPI
sasl.kerberos.service.name=kafka
```

Подключение пользователя к Kafka (создание тикетов) и работа с .sh файлами (скриптами)

1. Открыть терминальную сессию и подключиться к одному из брокеров Kafka.
2. Создать тикет для пользователя, введя пароль:

```
$ kinit -p admin-kafka@AD.RANGER-TEST


Password for admin-kafka@AD.RANGER-TEST:
```

> **ПРИМЕЧАНИЕ**
>
> - В данном примере `admin-kafka` - пользователь, имеющий запись в базе данных службы каталогов Active Directory и принципал для области `AD.RANGER-TEST`.
> - Полное описание функций команды kinit и применимых опций приведены по ссылке kinit.

3. Проверить тикет:

```
$ klist
```

```
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: admin-kafka@AD.RANGER-TEST

Valid starting     Expires            Service principal
08/19/2022 14:18:33  08/20/2022 00:18:33  krbtgt/AD.RANGER-TEST@AD.RANGER-TEST
        renew until 08/20/2022 14:17:35
```

4. Экспортируйте созданный файл client.jaas как параметр JVM для данного пользователя при помощи переменной среды `KAFKA_OPTS`:

```
$ export  KAFKA_OPTS="-Djava.security.auth.login.config=/tmp/client.jaas"
```

5. Создайте топик, указав путь к созданному файлу client.properties:

```
$ /usr/lib/kafka/bin/kafka-topics.sh --create --topic test-topic --bootstrap-server sov-ads-test-1.ru-central1.internal:9092,sov-ads-test-2.ru-central1.internal:9092,sov-ads-test-3.ru-central1.internal:9092 --command-config /tmp/client.properties
```

   Получите подтверждение:

```
Created topic test-topic.
```

6. Запишите сообщения в топик, указав путь к созданному файлу client.properties:

```
$ /usr/lib/kafka/bin/kafka-console-producer.sh --topic test-topic --bootstrap-server sov-ads-test-1.ru-central1.internal:9092,sov-ads-test-2.ru-central1.internal:9092,sov-ads-test-3.ru-central1.internal:9092 --producer.config /tmp/client.properties
```

```
>One
>Two
>Three
>Four
>Five
```

7. Считайте сообщения из топика, указав путь к созданному файлу client.properties:

```
$ /usr/lib/kafka/bin/kafka-console-consumer.sh --topic test-topic --from-beginning  --bootstrap-server sov-ads-test-1.ru-central1.internal:9092,sov-ads-test-2.ru-central1.internal:9092,sov-ads-test-3.ru-central1.internal:9092 --consumer.config /tmp/client.properties
```

   Полученные сообщения:

```
One
Two
Three
Four
Five
```

   Убедитесь в корректности принятых сообщений.

Защита канала по протоколу SSL в Kafka

Ольга Семме

Содержание

Назад к содержанию

Проверка настроек SSL

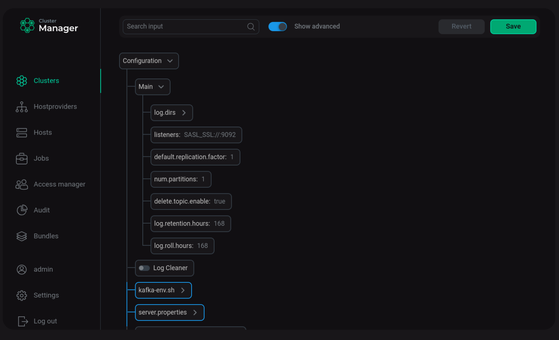
Подключение к Kafka (создание тикетов) и работа с .sh файлами (скриптами) с участием разных пользователей после установки SSL

ПРИМЕЧАНИЕ

Включение защиты канала про протоколу SSL осуществляется в соответствии со статьей Защита канала по протоколу SSL.

Проверка настроек SSL

1. Проверьте, что после запуска SSL в настройках сервиса Kafka в группе ***Main*** изменился параметр ***listeners*** с `SASL_PLAINTEXT://:9092` на `SASL_SSL://:9092`.



Настройки сервиса Kafka

2. Проверьте изменение файла *server.properties* на каждом сервере Kafka.
   Открыть файл *server.properties*:

```
$ sudo vim /usr/lib/kafka/config/server.properties
```

3. Убедитесь, что для каждого брокера Kafka строка, определяющая протокол безопасности, изменена на `SASL_SSL` и появились строки, определяющие хранилища для RSA-ключей серверов и пароли для них:

```
security.inter.broker.protocol=SASL_SSL
...
...
...
ssl.keystore.location=/tmp/keystore.jks
ssl.keystore.password=bigdata
ssl.key.password=bigdata
ssl.keystore.type=JKS
ssl.truststore.location=/tmp/truststore.jks
ssl.truststore.password=bigdata
ssl.truststore.type=JKS
```

Подключение к Kafka (создание тикетов) и работа с .sh файлами (скриптами) с участием разных пользователей после установки SSL

1. Для создания файла конфигурации *.properties* для пользователя с учетом SSL выполнить команду:

```
$ sudo vim /tmp/client_ssl.properties
```

   Заполнить файл данными:

```
security.protocol=SASL_SSL
sasl.mechanism=GSSAPI
sasl.kerberos.service.name=kafka
```

2. Выполнить шаги по подключению к Kafka под пользователем `writer` из статьи Использование MIT Kerberos в Kafka:
   - Открыть терминальную сессию 1 и подключиться к одному из брокеров Kafka.
   - Создать тикет для пользователя `writer`.
   - Проверить тикет.
   - Экспортировать файл *client.jaas* как параметр JVM для данного пользователя при помощи переменной среды `KAFKA_OPTS`.
3. Создать топик, указав путь к созданному файлу *client_ssl.properties*:

```
$ /usr/lib/kafka/bin/kafka-topics.sh --create --topic test-topic1 --bootstrap-server sov-ads-test-1.ru-central1.internal:9092,sov-ads-test-2.ru-central1.internal:9092,sov-ads-test-3.ru-central1.internal:9092 --command-config /tmp/client_ssl.properties
```

ПРИМЕЧАНИЕ

Файл *client.jaas* изменять после установки SSL не требуется.

4. Записать сообщение в топик:

```
$ /usr/lib/kafka/bin/kafka-console-producer.sh --topic test-topic1 --bootstrap-server sov-ads-test-1.ru-central1.internal:9092,sov-ads-test-2.ru-central1.internal:9092,sov-ads-test-3.ru-central1.internal:9092 --producer.config /tmp/client_ssl.properties
```

5. Выполнить шаги по подключению к Kafka под пользователем `reader` из статьи Использование MIT Kerberos в Kafka:
   - Открыть терминальную сессию 2 и подключиться к ней к одному из брокеров Kafka.
   - Создать тикет для пользователя `reader`.
   - Проверить тикет.
   - Экспортировать созданный файл *client.jaas* как параметр JVM для данного пользователя при помощи переменной среды `KAFKA_OPTS`.
6. Считать сообщения из топика, указав путь к созданному файлу *client_ssl.properties*:

```
$ /usr/lib/kafka/bin/kafka-console-consumer.sh --topic test-topic1 --from-beginning  --bootstrap-server sov-ads-test-1.ru-central1.internal:9092,sov-ads-test-2.ru-central1.internal:9092,sov-ads-test-3.ru-central1.internal:9092 --consumer.config /tmp/client_ssl.properties
```

   Убедиться, что сообщения считываются корректно.

Плагин Ranger Kafka

Ольга Семме

Содержание

Назад к содержанию

Предварительные требования

Активация плагина Ranger Kafka

Проверка активированного плагина Ranger Kafka

Предварительные требования

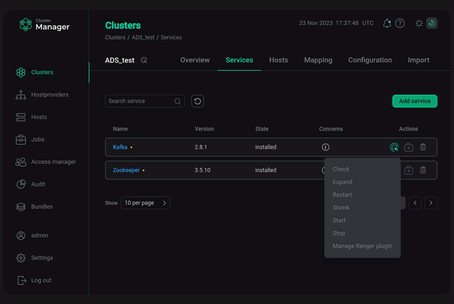
Предварительные требования для авторизации в Kafka при помощи Ranger:

1. Установлен кластер ADS с настроенной аутентификацией по протоколам MIT Kerberos или MS Active Directory.
2. Установлен кластер ADPS.
3. Включена интеграция с кластером ADS в кластере ADPS.

Активация плагина Ranger Kafka

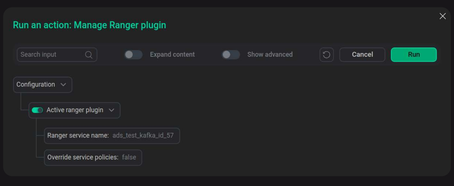
Для активации плагина Ranger Kafka:

1. Инициируйте активацию плагина Ranger Kafka. Для этого примените действие ***Manage Ranger plugin***, нажав на иконку в столбце ***Actions*** сервиса Kafka.



Активация плагина Ranger Kafka

В открывшемся окне ***Run an action: Manage Ranger plugin*** активируйте переключатель ***Active ranger plugin***. При необходимости установите название сервиса Ranger, который будет добавлен. Если сервис с таким именем уже существует, его можно пересоздать, активировав параметр ***Override service policies*** (`true` — параметр активен, `false` — неактивен), в таком случае старый сервис будет удален, а для нового сервиса будут заново сгенерированы политики. Если параметр ***Override service policies*** неактивен — после активации плагина осуществится подключение к существующему сервису Ranger.

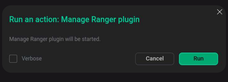


Включение плагина Ranger Kafka

ПРИМЕЧАНИЕ

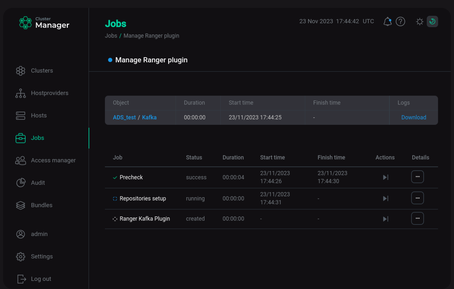
При первом включении плагина сервис с политиками будет создан (если еще не существует) вне зависимости от значения параметра ***Override service policies***.

2. Кликните ***Run*** и подтвердите действие в открывшемся окне.



Подтверждение действия

3. Дождитесь окончания процесса активации плагина Ranger Kafka и создания дефолтной политики на стороне Ranger. Проанализируйте и исправьте ошибки в случае их возникновения.

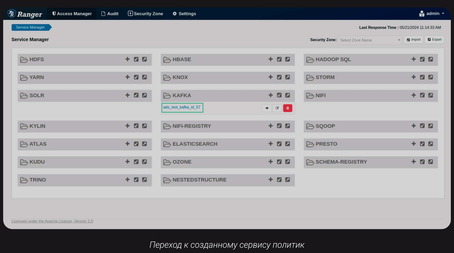


Процесс установки плагина Ranger Kafka

Проверка активированного плагина Ranger Kafka

После успешного включения в интерфейсе ***Service manager*** появляется сервис политик авторизации для сервиса Kafka выбранного кластера ADS.

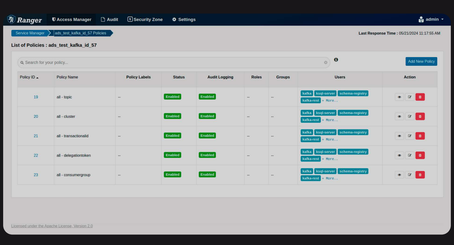
Для просмотра созданных политик для кластера нажмите на название сервиса Ranger для Kafka.



Переход к созданному сервису политик

При этом становятся доступны политики, автоматически созданные для серверов кластера.

Для просмотра, редактирования или удаления политик используются кнопки в столбце ***Action***.



Созданные политики для кластера ADS

Использование Ranger в Kafka

Ольга Синяк

Содержание

Назад к содержанию

Настройка политики авторизации

Работа авторизованного пользователя в Kafka

После активации плагина Ranger Kafka, создания сервиса Ranger или подключения к существующему сервису (в зависимости от значения параметра Override service policies) все действия в сервисе Kafka (создание топиков, запись сообщений, чтение сообщений, изменение конфигурации топиков и т.д.) возможны только после настройки политики авторизации для пользователя.

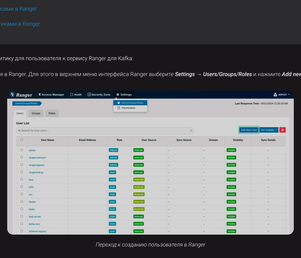
Настройка политики авторизации

ПРИМЕЧАНИЕ

- Перед настройкой авторизации для данного пользователя должен быть создан принципал в базе данных MIT Kerberos или учетная запись в базе данных MS Active Directory.
- Для получения дополнительной информации по авторизации в Kafka при помощи Ranger можно обратиться к статье Adding KAFKA Policies.
- Для получения дополнительной информации о работе с Ranger можно обратиться к следующим статьям:
  - Управление сервисами в Ranger
  - Управление политиками в Ranger

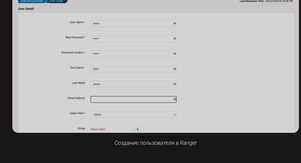
Чтобы добавить новую политику для пользователя к сервису Ranger для Kafka:

1. Создайте пользователя в Ranger. Для этого в верхнем меню интерфейса Ranger выберите *Settings* → *Users/Groups/Roles* и нажмите *Add new user*.



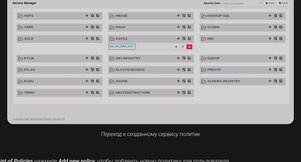
Переход к созданию пользователя в Ranger

2. В открывшемся окне заполните необходимые данные для пользователя:
   - *User Name* — имя пользователя. Это имя не может дублироваться в системе. Имя должно совпадать с именем пользователя в базе данных KDC или LDAP. Например, для пользователя с принципалом `writer@ADS-KAFKA.LOCAL` имя в Ranger должно быть `writer`.
   - *New Password* — пароль пользователя.
   - *Password Confirm* — подтверждение пароля пользователя.
   - *First Name* — личное имя пользователя.
   - *Last Name* — фамилия пользователя.
   - *Email Address* — email-адрес пользователя.
   - *Select Role* — выбор роли пользователя `Admin` `User` `Auditor`.
   - *Group* — выбор группы/групп, к которым принадлежит пользователь.



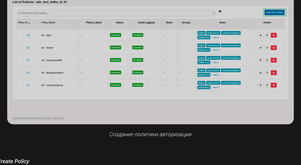
Создание пользователя в Ranger

3. Кликните *Save*.
4. В окне *Service Manager* кликните на название сервиса Ranger для Kafka.



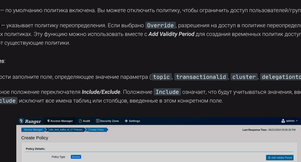
Переход к созданию сервису политик

5. В открывшемся окне *List of Policies* нажмите *Add new policy*, чтобы добавить новую политику для пользователя.

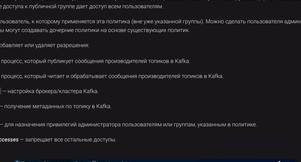


Создание политики авторизации

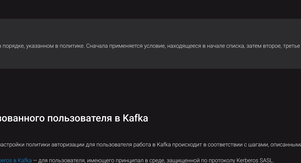
6. В открывшемся окне *Create Policy*:
   - заполните поля в разделе *Policy Details*:
     - *Policy Name* — имя политики. Это имя не может быть дублировано для такого же сервиса в системе. Поле обязательно для заполнения.
     - *Policy Label* — предоставляет следующие возможности:
       - Позволяет пользователю группировать наборы политик с помощью одной или нескольких меток.
       - Пользователь может искать политики по названиям меток. Поиск можно осуществлять как на странице списка политик, так и на странице отчета.
       - Помогает пользователю экспортировать/импортировать политики. Если пользователю необходимо экспортировать определенный набор политик, он может найти метку политики и экспортировать определенный набор политик.
     - *Description* — цель политики. Это поле является необязательным.
     - *Policy Conditions* — нажав в под *Add conditions*, можно добавить дополнительные условия.
     - *Add Validity Period* — время начала и окончания политики.



   - в разделе *Policy Details* установите нужные положения переключателей:
     - *Audit Logging* — определяет, будет ли проводиться аудит конкретной политики.
     - *Enable/Disabled* — по умолчанию политика включена. Вы можете отключить политику, чтобы ограничить доступ пользователей/групп для этой политики.
     - *Override/Normal* — указывает политику переопределения. Если выбрано `Override`, разрешения на доступ в политике переопределяют разрешения на доступ в существующих политиках. Эту функцию можно использовать вместе с *Add Validity Period* для создания временных политик доступа, которые переопределяют существующие политики.

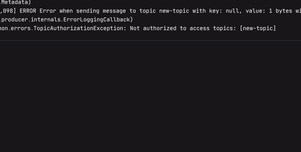


   - в разделе *Resources*:
     - При необходимости заполните поле, определяющее значение параметра `topic` `transactionalid` `cluster` `delegationtoken` `consumergroup`.
     - Установите нужное положение переключателя *Include/Exclude*. Положение `Include` означает, что будут учитываться значения, введенные в поле. Положение `Exclude` исключают все имена таблиц или столбцов, введенные в этом конкретном поле.



Разделы Policy Details и Resources окна Create Policy

   - заполните поля в разделе *Allow Condition*
     - *Select Role* — роль, к которой применяется эта политика. Роль — набор разрешений. Роли представляют собой более простой способ управления набором разрешений на основе определенных критериев доступа.
     - *Select Group* — группа, к которой применяется эта политика. Чтобы повысить статус пользователя до администратора, установите флажок *Delegate Admin*. Администраторы могут редактировать или удалять политику и создавать дочерние политики. Публичная группа содержит всех пользователей, поэтому предоставление доступа к публичной группе дает доступ всем пользователям.
     - *Select User* — пользователь, к которому применяется эта политика (вне уже указанной группы). Можно сделать пользователя администратором этой политики. Администраторы могут создавать дочерние политики на основе существующих политик.
     - *Permissions* — добавляет или удаляет разрешения:
       - `Publish` — процесс, который публикует сообщения производителей топиков в Kafka.
       - `Consume` — процесс, который читает и обрабатывает сообщения производителей топиков в Kafka.
       - `Configure` — настройка брокера/кластера Kafka.
       - `Describe` — получение метаданных по топику в Kafka.
     - *Delegate Admin* — для назначения привилегий администратора пользователям или группам, указанным в политике.
     - *Deny All Other Accesses* — запрещает все остальные доступы.



Раздел Allow Condition окна Create Policy

ПРИМЕЧАНИЕ

Условия выполняются в порядке, указанном в политике. Сначала применяется условие, находящееся в начале списка, затем второе, третье и так далее.

7. Кликните *Add*.

Работа авторизованного пользователя в Kafka

После активации Ranger и настройки политики авторизации для пользователя работа в Kafka происходит в соответствии с шагами, описанными в статьях:

- Использование MIT Kerberos в Kafka — для пользователя, имеющего принципал в среде, защищенной по протоколу Kerberos SASL.
- Защита канала по протоколу SSL в Kafka — для пользователя, имеющего принципал в среде, использующей протокол Kerberos SASL с защитой канала по протоколу SSL.
- Использование Kerberos с MS Active Directory в Kafka — для пользователя, имеющего учетную запись в Active Directory и соответствующий сертификат SSL LDAP-сервера.

Любое действие неавторизованного пользователя вызовет ошибку. Например, попытка записать сообщения в топик `new-topic` для пользователя, в политике авторизации которого не указано разрешение `Publish`, вызовет ошибки следующего содержания:

```
[2022-08-29 10:21:10,094] WARN [Producer clientId=console-producer] Error while fetching metadata with correlation id 3 : {new-topic=TOPIC_AUTHORIZATION_FAILED} (org.apache.kafka.clients.NetworkClient)
[2022-08-29 10:21:10,097] ERROR [Producer clientId=console-producer] Topic authorization failed for topics [new-topic] (org.apache.kafka.clients.Metadata)
[2022-08-29 10:21:10,098] ERROR Error when sending message to topic new-topic with key: null, value: 1 bytes with error: (org.apache.kafka.clients.producer.internals.ErrorLoggingCallback)
org.apache.kafka.common.errors.TopicAuthorizationException: Not authorized to access topics: [new-topic]
```

Содержание

Назад к содержанию

Включение ACL в Kafka
Подготовка среды аутентификации к созданию списков авторизации ACL
SASL_PLAINTEXT
Kerberos SASL
Конфигурация списков ACL
Назначение прав и запретов на отдельные действия в Kafka для пользователей
Назначение прав producer и consumer пользователю
Просмотр списков ACL
Работа авторизованного пользователя в Kafka

В данной статье описаны основные действия для работы с авторизацией ACL в Kafka.

Access Control List (ACL) – список управления доступом, который определяет, кто может получать доступ к топикам или группам в Kafka и какие именно операции разрешено или запрещено выполнять пользователям или группе пользователей.

Kafka имеет **AclAuthorizer** – встроенный авторизатор на базе ACL.

Списки авторизации ACL для Kafka хранятся в ZooKeeper.

Включение ACL в Kafka

ПРИМЕЧАНИЕ

Перед настройкой авторизации для кластера ADS должна быть включена и настроена аутентификация и созданы пользователи при помощи одного из протоколов:

- SASL_PLAINTEXT
- SAT Kerberos
- Kerberos с MS Active Directory

Чтобы включить и использовать AclAuthorizer, необходимо выполнить:

1. На всех в брокерах Kafka ввести команду:

```
$ sudo vim /usr/lib/kafka/config/server.properties
```

2. Добавить в файл новые параметры:

```
authorizer.class.name=kafka.security.authorizer.AclAuthorizer
allow.everyone.if.no.acl.found=true
```

где

- `authorizer.class.name` – установка класса авторизатора
- `allow.everyone.if.no.acl.found` – предоставлять всем права по авторизации только для топиков, внесенных в список ACL.

3. После внесения данных рестартовать сервис Kafka на всех хостах с брокерами Kafka и проверить статус сервиса. Для этого поочередно ввести команды:

```
$ sudo systemctl restart kafka
$ sudo systemctl status kafka
```

В результате выведется информация о статусе сервиса Kafka:

```
kafka.service - Apache Kafka
   Loaded: loaded (/usr/lib/systemd/system/kafka.service; enabled; vendor preset: disabled)
  Drop-In: /etc/systemd/system/kafka.service.d
           └─custom.conf
   Active: active (running) since Wed 2022-10-12 12:48:49 UTC; 32s ago
  Process: 8792 ExecStop=/usr/lib/kafka/bin/kafka-stop (code=exited, status=0/SUCCESS)
  Process: 8857 ExecStart=/usr/lib/kafka/bin/kafka-start (code=exited, status=0/SUCCESS)
  Process: 8854 ExecStartPre=/bin/chown kafka:kafka /run/kafka (code=exited, status=0/SUCCESS)
  Process: 8852 ExecStartPre=/bin/mkdir -p /run/kafka (code=exited, status=0/SUCCESS)
 Main PID: 8860 (java)
    Tasks: 82
   Memory: 298.6M
   CGroup: /system.slice/kafka.service
           └─8860 java -Xmx1G -Xms1G -server -XX:+UseG1GC -XX:MaxGCPauseMillis=20 -XX:InitiatingHeapOcc...

Oct 12 12:48:49 sov-ads-test-1.ru-central1.internal systemd[1]: Starting Apache Kafka...
Oct 12 12:48:49 sov-ads-test-1.ru-central1.internal systemd[1]: Started Apache Kafka.
```

ВНИМАНИЕ

Перезапуск сервиса Kafka при помощи средств пользовательского интерфейса ADCM может привести к удалению установленного значения `authorizer.class.name`

Подготовка среды аутентификации к созданию списков авторизации ACL

SASL_PLAINTEXT

Для создания среды пользователя, указанного при аутентификации по протоколу SASL_PLAINTEXT, необходимо:

1. На всех в брокерах Kafka ввести команду:

```
$ sudo vim /etc/kafka/conf/client.properties
```

2. В открывшемся файле включить (раскомментировать) механизм аутентификации и заполнить данные для созданного пользователя:

```
security.protocol=SASL_PLAINTEXT
sasl.mechanism=PLAIN
# Uncomment and set necessary username/password
#sasl.jaas.config=org.apache.kafka.common.security.plain.PlainLoginModule required
#   username="client" \
#   password="client-secret";
```

Дополнительные сведения о работе с пользователями в среде аутентификации SASL_PLAINTEXT можно найти в статье «Использование SASL_PLAINTEXT в Kafka».

Kerberos SASL

Для создания среды пользователя, указанного при аутентификации по протоколу Kerberos SASL, необходимо:

1. Создать JAAS-файл (Java Authentication and Authorization Service) для принципала `user`, управляющего списками ACL. Выполнить команду:

```
$ sudo vim /tmp/client.jaas
```

2. Внести данные в файл:

```
KafkaClient {
com.sun.security.auth.module.Krb5LoginModule required
useTicketCache=true;
};
```

3. Создать тикет для пользователя `user`:

```
$ kinit -p user@ADS-KAFKA.LOCAL
```

4. Проверить тикет:

```
$ klist
```

```
Ticket cache: FILE:/tmp/krb5cc_1000
Default principal: user@ADS-KAFKA.LOCAL

Valid starting     Expires            Service principal
08/18/2022 20:44:12  08/11/2022 20:44:12  krbtgt/ADS-KAFKA.LOCAL@ADS-KAFKA.LOCAL
```

5. Экспортировать созданный файл client.jaas как параметр JVM для данного пользователя при помощи переменной среды `KAFKA_OPTS`:

```
$ export  KAFKA_OPTS="-Djava.security.auth.login.config=/tmp/client.jaas"
```

Дополнительные сведения о работе с пользователями в среде аутентификации Kerberos SASL можно найти в статье «Использование SAT Kerberos в Kafka».

Конфигурация списков ACL

Работа с авторизацией на основе ACL происходит путем запуска скрипта kafka-acls.sh в командной строке с любого узла кластера, на котором установлен сервис Kafka.

ПРИМЕЧАНИЕ

Для просмотра всех опций для скрипта kafka-acls.sh необходимо выполнить команду:

```
$ /usr/lib/kafka/bin/kafka-acls.sh
```

Назначение прав и запретов на отдельные действия в Kafka для пользователей

Назначение прав или запретов на отдельные действия пользователю или группе при помощи скрипта kafka-acls.sh с указанием опции `--operation` и операции, на которую дается доступ.

Назначение прав или запретов осуществляется при помощи опции `--add`.

Опции `--allow-principal` – разрешает доступ, а `--deny-principal` – запрещает действие.

Удаление пользователя из списка ACL выполняется при помощи опции `--remove`.

Основные операции, для которых можно установить права пользователю:

- `Describe` – просмотр метаданных топика или группы.
- `DescribeConfigs` – просмотр конфигурации топика.
- `Alter` – изменение топика или группы.
- `IdempotentWrite` – запись в топик для идемпотентных и транзакционных API.
- `Read` – право на чтение из топика или группы.
- `Delete` – удаление топика или группы.
- `Create` – создание топика или группы.
- `All` – все права.
- `Write` – право на запись в топик или группу.
- `AlterConfigs` – изменение конфигурации топика.

ПРИМЕЧАНИЕ

Для пользователей, прошедших аутентификацию по протоколу Kerberos SASL или LDAP, в составлении списков ACL используется имя пользователя без обозначения области аутентификации (realm).

Для того, чтобы разрешить пользователю: `username` выполнить операцию: `WRITE` над топиком: `test-topic`, необходимо на хосте с брокером Kafka ввести команду:

```
$ /usr/lib/kafka/bin/kafka-acls.sh --bootstrap-server localhost:9092 --command-config /etc/kafka/conf/client.properties --add --allow-principal User:username --operation WRITE --test-topic
```

Для того, чтобы запретить пользователю: `username1` выполнить операцию: `WRITE` над топиком: `test-topic`, необходимо на хосте с брокером Kafka ввести команду:

```
$ /usr/lib/kafka/bin/kafka-acls.sh --bootstrap-server localhost:9092 --command-config /etc/kafka/conf/client.properties --add --deny-principal User:username1 --operation WRITE --test-topic
```

В результате каждого добавления в список появляется сообщение об успешном добавлении того или запрета, а также список ACL с текущими правами пользователей для топика:

```
Adding ACLs for resource `ResourcePattern(resourceType=TOPIC, name=test-topic, patternType=LITERAL)`:
        (principal=User:username, host=*, operation=WRITE, permissionType=ALLOW)

Current ACLs for resource `ResourcePattern(resourceType=TOPIC, name=test-topic, patternType=LITERAL)`:
        (principal=User:username, host=*, operation=WRITE, permissionType=ALLOW)
```

Назначение прав producer и consumer пользователю

Существует возможность назначить пользователю сразу несколько прав производителя (producer) – создание (create), запись (write), просмотр метаданных (describe).

Чтобы добавить права `producer` для топика `test-topic` пользователю: `writer`, необходимо на хосте с брокером Kafka ввести команду:

```
$ /usr/lib/kafka/bin/kafka-acls.sh --bootstrap-server localhost:9092 --command-config /etc/kafka/conf/client.properties  --add --allow-principal User:writer --producer --topic test-topic
```

В результате появляется сообщение об успешном добавлении прав `producer` и список ACL с текущими правами пользователей для топика:

```
Adding ACLs for resource `ResourcePattern(resourceType=TOPIC, name=test-topic, patternType=LITERAL)`:
        (principal=User:writer, host=*, operation=CREATE, permissionType=ALLOW)
        (principal=User:writer, host=*, operation=WRITE, permissionType=ALLOW)
        (principal=User:writer, host=*, operation=DESCRIBE, permissionType=ALLOW)

Current ACLs for resource `ResourcePattern(resourceType=TOPIC, name=test-topic, patternType=LITERAL)`:
        (principal=User:writer, host=*, operation=CREATE, permissionType=ALLOW)
        (principal=User:writer, host=*, operation=WRITE, permissionType=ALLOW)
        (principal=User:writer, host=*, operation=DESCRIBE, permissionType=ALLOW)
```

Также существует возможность назначить сразу несколько прав потребителя (consumer) – чтение (read), просмотр метаданных (describe).

Чтобы добавить права `consumer` для топика `test-topic` пользователю: `reader`, необходимо на хосте с брокером Kafka ввести команду:

```
$ /usr/lib/kafka/bin/kafka-acls.sh --bootstrap-server localhost:9092 --command-config /etc/kafka/conf/client.properties  --add --allow-principal User:reader --consumer --topic test-topic --group 1
```

ПРИМЕЧАНИЕ

При описании прав потребителя (consumer) необходимо указать и название топика (`topic`), и название группы потребителей (`group`), даже если потребитель всего один.

В результате появляется сообщение об успешном добавлении прав `consumer` и список ACL с текущими правами пользователей для топика `test-topic`, включая созданные ранее для пользователя `writer`:

```
Adding ACLs for resource `ResourcePattern(resourceType=TOPIC, name=test-topic, patternType=LITERAL)`:
        (principal=User:reader, host=*, operation=DESCRIBE, permissionType=ALLOW)
        (principal=User:reader, host=*, operation=READ, permissionType=ALLOW)

Current ACLs for resource `ResourcePattern(resourceType=TOPIC, name=test-topic, patternType=LITERAL)`:
        (principal=User:writer, host=*, operation=CREATE, permissionType=ALLOW)
        (principal=User:writer, host=*, operation=WRITE, permissionType=ALLOW)
        (principal=User:writer, host=*, operation=DESCRIBE, permissionType=ALLOW)
        (principal=User:reader, host=*, operation=DESCRIBE, permissionType=ALLOW)
        (principal=User:reader, host=*, operation=READ, permissionType=ALLOW)
```

Также выведется данные о созданных правах для группы потребителей:

```
Current ACLs for resource `ResourcePattern(resourceType=GROUP, name=1, patternType=LITERAL)`:
        (principal=User:reader, host=*, operation=DESCRIBE, permissionType=ALLOW)
        (principal=User:reader, host=*, operation=READ, permissionType=ALLOW)
```

Просмотр списков ACL

Просмотр списков ACL осуществляется при помощи опции `--list`. Для того, чтобы просмотреть список прав для топика `test-topic`, необходимо выполнить команду:

```
$ /usr/lib/kafka/bin/kafka-acls.sh --bootstrap-server localhost:9092 --command-config /etc/kafka/conf/client.properties --list --topic test-topic
```

В результате выведется список всех назначенных прав для топика `test-topic`:

```
Current ACLs for resource `ResourcePattern(resourceType=TOPIC, name=test-topic, patternType=LITERAL)`:
        (principal=User:writer, host=*, operation=CREATE, permissionType=ALLOW)
        (principal=User:writer, host=*, operation=WRITE, permissionType=ALLOW)
        (principal=User:writer, host=*, operation=DESCRIBE, permissionType=ALLOW)
        (principal=User:reader, host=*, operation=DESCRIBE, permissionType=ALLOW)
        (principal=User:reader, host=*, operation=READ, permissionType=ALLOW)
```

Для просмотра списков прав для всех топиков используется такая же команда без указания имени топика.

Работа авторизованного пользователя в Kafka

После настройки списков ACL для пользователей работа в Kafka происходит в соответствии с шагами, описанными в статьях:

- «Использование SAT Kerberos в Kafka» – для пользователя, имеющего принципал в среде, защищенной по протоколу Kerberos SASL.
- «Защита канала по протоколу SSL в Kafka» – для пользователя, имеющего принципал в среде, использующей протокол Kerberos SASL с защитой канала по протоколу SSL.
- «Использование Kerberos с MS Active Directory в Kafka» – для пользователя, имеющего учетную запись в Active Directory и сертификацией сертификат SSL LDAP-сервера.

Любое действие неавторизованного пользователя вызывает ошибку. Например, попытка записать сообщение в топик `test-topic` для пользователя, не имеющего разрешение `WRITE` в списке ACL для данного топика, вызовет ошибку следующего содержания:

```
[2022-10-12 13:38:57,261] WARN [Producer clientId=console-producer] Error while fetching metadata with correlation id 9 : {test-topic=TOPIC_AUTHORIZATION_FAILED} (org.apache.kafka.clients.NetworkClient)
[2022-10-12 13:38:57,265] ERROR [Producer clientId=console-producer] Topic authorization failed for topics [test-topic] (org.apache.kafka.clients.Metadata)
[2022-10-12 13:38:57,267] ERROR Error when sending message to topic test-topic with key: null, value: 1 bytes with error: (org.apache.kafka.clients.producer.internals.ErrorLoggingCallback)
org.apache.kafka.common.errors.TopicAuthorizationException: Not authorized to access topics: [test-topic]
```

Создание приложения Kafka Streams

Илья Космодемьянский

Вход в подписки

Содержание

- Подготовка к созданию проекта
  - Установка OpenJDK
  - Установка последней версии Apache Maven
  - Настройка переменных среды
  - Проверка установки Apache Maven
- Создание проекта Kafka Streams
  - Создание структуры проекта
  - Описание структуры проекта
  - Настройка pom.xml
  - Готовые приложения Kafka Streams
- Запуск и работа с приложениями Kafka Streams
  - Kpp_Pipe.java
  - Запуск приложения
  - Проверка работы созданного приложения

В этом статье создадим простое приложение на потоковой обработке с использованием Kafka Streams.

Kafka Streams — клиентская библиотека для потоковой обработки данных, которая использует возможности Apache Kafka.

Для создания приложения Kafka Streams используется [Streams DSL](#) — интерфейс управления потоками приложения.

Для примера создадим приложение, которое будет считать количество слов в потоке данных, поступающих в Apache Kafka.

Подготовка к созданию проекта

Apache Kafka написан на Java и Scala, поэтому для работы с ним потребуется установленная виртуальная машина Java.

В данном примере будет использоваться OpenJDK 11.

Установка OpenJDK

OpenJDK используется по умолчанию в большинстве дистрибутивов Linux.

Проверить наличие OpenJDK 11 можно командой `yum list java-11-openjdk-devel` (для RHEL, CentOS, Fedora) или `apt list openjdk-11-jdk` (для Debian, Ubuntu).

Если OpenJDK 11 не установлен, установить его можно командой `yum install java-11-openjdk-devel` (для RHEL, CentOS, Fedora) или `apt install openjdk-11-jdk` (для Debian, Ubuntu).

Для установки OpenJDK необходимо выполнить команду:

```
sudo yum install java-1.8.0-openjdk
```

Результат:

```
...
Complete!
```

Проверим установленную версию Java, выполнив следующую команду:

```
java -version
```

При успешной установке должно отобразиться:

```
openjdk version "1.8.0_242"
OpenJDK Runtime Environment (build 1.8.0_242-b12)
OpenJDK 64-Bit Server VM (build 25.242-b12, mixed mode)
```

Установка последней версии Apache Maven

Apache Maven — инструмент для автоматической сборки проектов, написанных на Java. Для его установки выполним команду `wget` для загрузки архива Apache Maven и команду `tar` для распаковки архива:

```
wget http://ftp.time.mk.ua/apache/maven/maven-3/3.6.3/binaries/apache-maven-3.6.3-bin.tar.gz -P /tmp
```

Результат:

```
...
2020-03-31 14:03:27 (1,01 MB/s) - '/tmp/apache-maven-3.6.3-bin.tar.gz' saved [8913638/8913638]
```

Распакуем архив в каталог `/opt`:

```
sudo tar xf /tmp/apache-maven-3.6.3-bin.tar.gz -C /opt
```

Для удобства установим символическую ссылку на каталог с Maven, чтобы при обновлении версии Maven не изменять переменные среды:

```
sudo ln -s /opt/apache-maven-3.6.3 /opt/maven
```

Настройка переменных среды

Для обращения к Java и устанавливаемому Apache Maven необходимо добавить путь к каталогу с исполняемыми файлами Java и Maven в переменную среды `PATH`. Это можно сделать несколькими способами.

1. Создать файл скрипта

```
sudo vim /etc/profile.d/maven.sh
```

2. Записать в него следующее, сохранить и закрыть:

```
export JAVA_HOME=/usr/lib/jvm/jre-openjdk
export M2_HOME=/opt/maven
export MAVEN_HOME=/opt/maven
export PATH=${M2_HOME}/bin:${PATH}
```

3. Сделать скрипт исполняемым:

```
sudo chmod +x /etc/profile.d/maven.sh
```

4. Для применения изменений выполнить:

```
sudo source /etc/profile.d/maven.sh
```

5. Выполнить один раз команду `source`:

```
source /etc/profile.d/maven.sh
```

Проверка установки Apache Maven

Для проверки установки Apache Maven выполним команду:

```
mvn -version
```

Результат:

```
Apache Maven 3.6.3 (cecedd343f7ac7bd319b514b0d44968e743f0bb4)
Maven Home: /opt/maven
Java version: 1.8.0_242, vendor: Red Hat, Inc., runtime: /usr/lib/jvm/java-1.8.0-openjdk-1.8.0.242.b08-1.el7.x86_64/jre
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "3.10.0-1062.12.1.el7.x86_64", arch: "amd64", family: "unix"
```

Создание проекта Kafka Streams

Создание структуры проекта

В нашем Maven-проекте нам потребуется создать несколько классов. Для создания структуры приложения воспользуемся архетипом Quickstart.

Создадим структуру проекта Kafka Streams в каталоге. Для Apache Maven выполним команду:

```
mvn archetype:generate \
    -DarchetypeGroupId=org.apache.kafka \
    -DarchetypeArtifactId=streams-quickstart-java \
    -DarchetypeVersion=2.4.1 \
    -DgroupId=streams.examples \
    -DartifactId=streams.examples \
    -Dversion=0.1 \
    -Dpackage=myapps
```

Здесь используются следующие параметры:

- `archetype:generate` — стартовая архитектурная заготовка проекта
- `archetypeGroupId` — группа для архитектурной заготовки
- `archetypeArtifactId` — название используемого архетипа
- `archetypeVersion` — версия используемого архетипа

Значения параметров `groupId`, `artifactId` и `version` можно изменить.

- `groupId` — группа создаваемых ПО, уникальное для организации. Обычно здесь устанавливают имя корневого пакета
- `artifactId` — название создаваемого артефакта
- `version` — версия готовой создаваемого артефакта
- `package` — название главного java-пакета проекта

Результат выполнения команды `mvn archetype:generate` будет следующим образом:

```
...
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 14.917s
[INFO] Finished at: Tue Mar 31 14:31:41 UTC 2020
[INFO] Final Memory: 21M/165M
[INFO] ------------------------------------------------------------------------
```

Описание структуры проекта

Перейдем в каталог `streams.examples` и посмотрим его содержимое:

```
cd streams.examples
ls -la
```

Структура данных создаваемого проекта будет следующей:

```
.
├── pom.xml
├── src
│   └── main
│       ├── java
│       │   └── myapps
│       │       ├── App.java
│       │       ├── LineSplit.java
│       │       └── Pipe.java
│       └── resources
│           └── log4j.properties
```

Настройка pom.xml

В качестве системы управления зависимостями в проекте используется [Apache Maven](#). XML-файл, содержащий конфигурацию и правила сборки нашего проекта, называется pom.xml.

Откроем pom.xml в текстовом редакторе:

```
vim pom.xml
```

pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>streams.examples</groupId>
    <artifactId>streams.examples</artifactId>
    <version>0.1</version>
    <packaging>jar</packaging>

    <name>streams.examples</name>
    <url>http://www.example.com</url>

    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <kafka.version>2.4.0</kafka.version>
        <slf4j.version>1.7.30</slf4j.version>
        <log4j.version>1.2.17</log4j.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.apache.kafka</groupId>
            <artifactId>kafka-streams</artifactId>
            <version>${kafka.version}</version>
        </dependency>
        <dependency>
            <groupId>org.slf4j</groupId>
            <artifactId>slf4j-simple</artifactId>
            <version>${slf4j.version}</version>
        </dependency>
        <dependency>
            <groupId>log4j</groupId>
            <artifactId>log4j</artifactId>
            <version>${log4j.version}</version>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <artifactId>maven-compiler-plugin</artifactId>
                <configuration>
                    <source>1.8</source>
                    <target>1.8</target>
                </configuration>
            </plugin>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-shade-plugin</artifactId>
                <version>3.1.1</version>
                <executions>
                    <execution>
                        <phase>package</phase>
                        <goals>
                            <goal>shade</goal>
                        </goals>
                        <configuration>
                            <createDependencyReducedPom>false</createDependencyReducedPom>
                            <filters>
                                <filter>
                                    <artifact>*:*</artifact>
                                    <excludes>
                                        <exclude>META-INF/*.SF</exclude>
                                        <exclude>META-INF/*.DSA</exclude>
                                        <exclude>META-INF/*.RSA</exclude>
                                    </excludes>
                                </filter>
                            </filters>
                            <transformers>
                                <transformer implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
                                    <mainClass>myapps.App</mainClass>
                                </transformer>
                            </transformers>
                        </configuration>
                    </execution>
                </executions>
            </plugin>
        </plugins>
    </build>
</project>
```

Здесь используются следующие теги:

- `project` — корневой элемент pom.xml
- `modelVersion` — версия используемого POM. Для двух последних обязательно должно быть значение 4.0.0
- `groupId` — параметр, аналогичный параметру `groupId` при создании архетипа
- `name` — параметр, аналогичный параметру `artifactId` при создании архетипа
- `url` — ссылка на сайт проекта
- `properties` — свойства, которые можно использовать в других местах pom.xml. В данном случае они используются для задания версий библиотек
- `dependencies` — описание зависимостей проекта. Для каждой зависимости указывается `groupId`, `artifactId` и `version`
- `build` — описание процесса сборки проекта. Здесь мы задаем версию компилятора Java и настраиваем создание fat-jar

Готовые приложения Kafka Streams

После создания проекта в папке `src/main/java` находятся три готовых примера использования Kafka Streams:

- `App.java` — простое приложение, которое читает из топика `streams-plaintext-input` и записывает слова в топик `streams-linesplit-output`
- `LineSplit.java` — приложение, которое читает из топика `streams-plaintext-input` и записывает каждое слово в отдельное сообщение в топик `streams-linesplit-output`
- `Pipe.java` — приложение, которое читает из топика `streams-plaintext-input` и записывает данные в топик `streams-pipe-output`

В следующих шагах для примера используем приложение Pipe.java, а также покажем, как можно проверить работоспособность и корректность приложения.

Для работы с Kafka Streams нам потребуется создать топики `streams-plaintext-input` и `streams-pipe-output`. Создадим их командами:

```
/opt/kafka/bin/kafka-topics.sh --create --topic streams-plaintext-input --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1
/opt/kafka/bin/kafka-topics.sh --create --topic streams-pipe-output --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1
```

Запуск и работа с приложениями Kafka Streams

Kpp_Pipe.java

Чтобы использовать приложение Pipe.java, нужно в файле `src/main/java/myapps/Pipe.java` заменить `streams-plaintext-input` на `streams-pipe-output`.

Pipe

```
package myapps;

import org.apache.kafka.common.serialization.Serdes;
import org.apache.kafka.streams.KafkaStreams;
import org.apache.kafka.streams.StreamsBuilder;
import org.apache.kafka.streams.StreamsConfig;
import org.apache.kafka.streams.Topology;

import java.util.Properties;
import java.util.concurrent.CountDownLatch;

public class Pipe {

    public static void main(String[] args) throws Exception {
        Properties props = new Properties();
        props.put(StreamsConfig.APPLICATION_ID_CONFIG, "streams-pipe");
        props.put(StreamsConfig.BOOTSTRAP_SERVERS_CONFIG, "localhost:9092");
        props.put(StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG, Serdes.ByteArray().getClass());
        props.put(StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG, Serdes.ByteArray().getClass());

        final StreamsBuilder builder = new StreamsBuilder();

        builder.stream("streams-plaintext-input").to("streams-pipe-output");

        final Topology topology = builder.build();
        System.out.println(topology.describe());
        final KafkaStreams streams = new KafkaStreams(topology, props);
        final CountDownLatch latch = new CountDownLatch(1);

        Runtime.getRuntime().addShutdownHook(new Thread("streams-shutdown-hook") {
            @Override
            public void run() {
                streams.close();
                latch.countDown();
            }
        });

        try {
            streams.start();
            latch.await();
        } catch (Throwable e) {
            System.exit(1);
        }
        System.exit(0);
    }
}
```

В коде Pipe.java используются:

- `props.put` — установка различных параметров (подробнее [здесь](#))
  - `StreamsConfig.APPLICATION_ID_CONFIG` — имя приложения
  - `StreamsConfig.BOOTSTRAP_SERVERS_CONFIG` — адрес брокера
  - `StreamsConfig.DEFAULT_KEY_SERDE_CLASS_CONFIG` — класс, используемый для сериализации ключей. Здесь `StreamsConfig.getClass()` вызывается для получения `Class` объекта, `Serdes.ByteArray()` — создание экземпляра Serdes, объявленный "оригинальным" классом
  - `StreamsConfig.DEFAULT_VALUE_SERDE_CLASS_CONFIG` — класс, используемый для сериализации значений. Здесь `StreamsConfig.getClass()` вызывается для получения `Class` объекта, `Serdes.ByteArray()` — создание экземпляра Serdes, объявленный "оригинальным" классом
- `final StreamsBuilder builder = new StreamsBuilder()` — объявление топологии потока данных
- `builder.stream("streams-plaintext-input").to("streams-pipe-output")` — создание топологии потока данных, которая `streams-plaintext-input` перенаправляет в `streams-pipe-output`
- `final Topology topology = builder.build()` — создание топологии потока данных, объявленной "оригинально" (класс)
- `final KafkaStreams streams = new KafkaStreams(topology, props)` — создание потока данных, использующего топологию потока данных `topology` и настройки `props`
- `final CountDownLatch latch = new CountDownLatch(1)` — создание объекта `CountDownLatch`, для синхронизации потоков (`Thread`)
- `Runtime.getRuntime().addShutdownHook(new Thread("streams-shutdown-hook") {` — создание нового потока `streams-shutdown-hook`, который будет выполнен при завершении работы приложения
- `streams.start()` — запуск потока данных
- `latch.await()` — ожидание завершения работы потока данных
- `System.exit(0)` — завершение работы приложения

Запуск приложения

Приложение запускается командой `mvn exec:java`. Для удобства, запустим приложение в фоновом режиме:

```
mvn exec:java &
```

Результат:

```
[INFO] Building jar: /opt/maven/streams.examples/target/streams.examples-0.1.jar
[INFO] >>> exec-maven-plugin:3.0.0:java (default-cli) > validate @ streams.examples >>>
[INFO] maven-exec-plugin:3.0.0:java (default-cli) @ streams.examples ---
[INFO] Topologies:
[INFO]   Sub-topology: 0
[INFO]     Source: KSTREAM-SOURCE-0000000000 (topics: [streams-plaintext-input])
[INFO]       --> KSTREAM-SINK-0000000001
[INFO]     Sink: KSTREAM-SINK-0000000001 (topics: streams-pipe-output)
[INFO]       <-- KSTREAM-SOURCE-0000000000
```

Приложение запущено. Выполнение топологии потока данных приложения `Pipe.java`:

```
Topologies:
  Sub-topology: 0
    Source: KSTREAM-SOURCE-0000000000 (topics: [streams-plaintext-input])
      --> KSTREAM-SINK-0000000001
    Sink: KSTREAM-SINK-0000000001 (topics: streams-pipe-output)
      <-- KSTREAM-SOURCE-0000000000
```

Проверка работы созданного приложения

Для проверки работоспособности приложения необходимо открыть две дополнительные терминальные сессии.

1. Для создания тестовых сообщений в топике `streams-plaintext-input` запустим консольный продюсер:

```
/opt/kafka/bin/kafka-topics.sh --create --topic streams-plaintext-input --bootstrap-server localhost:9092 --partitions1 --replication-factor 1
```

2. Создадим консольный потребитель для топика `streams-pipe-output`:

```
/opt/kafka/bin/kafka-topics.sh --create --topic streams-pipe-output --bootstrap-server localhost:9092 --partitions1 --replication-factor 1
```

3. В другой консоли создать потребителя сообщений из топика `streams-pipe-output`, используя команду:

```
/opt/kafka/bin/kafka-console-consumer.sh --topic streams-pipe-output --from-beginning --bootstrap-server localhost:9092
```

4. Записывать сообщения в консоли производителя для топика `streams-plaintext-input` и читать полученные сообщения в консоли потребителя для топика `streams-pipe-output`.

Пример использования Kafka API

Содержание

- Подготовка к созданию приложения
  - Создание структуры проекта
  - Файловая структура
  - Подключение зависимостей
  - Настройка логирования
- Producer API
- Пример записи Producer
- Consumer API
- Пример чтения Consumer
- Admin API
- Пример работы Admin API

Перейти к приложению

Kafka — распределённый брокер сообщений, который обеспечивает надёжную доставку данных между системами. Для работы с ним используются специальные клиентские библиотеки, предоставляющие API для производителей, потребителей и административных операций.

В этом руководстве мы рассмотрим, как использовать Kafka API на практике. Мы создадим простое приложение, которое будет отправлять и получать сообщения, а также управлять топиками.

Для работы с Kafka API нам потребуется:

- Java 11 или выше
- Apache Kafka 3.x (скачать с официального сайта)
- Gradle 7.x или Maven 3.x для управления зависимостями

Мы будем использовать Gradle в качестве системы сборки.

Подготовка к созданию приложения

Перед написанием кода нам нужно настроить окружение. Создадим структуру проекта и добавим необходимые зависимости.

Создание структуры проекта

Файловая структура

```
kafka-example/
├── build.gradle
├── settings.gradle
├── src/
│   ├── main/
│   │   ├── java/
│   │   │   └── com/example/kafka/
│   │   │       ├── producer/
│   │   │       │   └── MessageProducer.java
│   │   │       ├── consumer/
│   │   │       │   └── MessageConsumer.java
│   │   │       ├── admin/
│   │   │       │   └── TopicManager.java
│   │   │       └── Application.java
│   │   └── resources/
│   │       └── application.properties
│   └── test/
│       └── java/
│           └── com/example/kafka/
└── README.md
```

Подключение зависимостей

Файл `build.gradle` для Gradle:

```
plugins {
    id 'java'
    id 'application'
}

group = 'com.example'
version = '1.0-SNAPSHOT'

repositories {
    mavenCentral()
}

dependencies {
    implementation 'org.apache.kafka:kafka-clients:3.6.1'
    implementation 'org.slf4j:slf4j-api:2.0.9'
    implementation 'ch.qos.logback:logback-classic:1.4.14'
    testImplementation 'org.junit.jupiter:junit-jupiter:5.10.1'
}

application {
    mainClass = 'com.example.kafka.Application'
}

test {
    useJUnitPlatform()
}
```

Настройка логирования

Файл `src/main/resources/logback.xml`:

```
<configuration>
    <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
        <encoder>
            <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>
        </encoder>
    </appender>

    <logger name="org.apache.kafka" level="WARN" />
    <logger name="com.example.kafka" level="DEBUG" />

    <root level="INFO">
        <appender-ref ref="STDOUT" />
    </root>
</configuration>
```

Producer API

В этом разделе мы рассмотрим, как создать производителя сообщений, который будет отправлять данные в Kafka. Producer API позволяет асинхронно отправлять сообщения в топики, а также получать подтверждения о доставке.

Конфигурация Producer

```
# Kafka broker addresses
bootstrap.servers=localhost:9092

# Producer settings
acks=all
retries=3
batch.size=16384
linger.ms=1
buffer.memory=33554432
key.serializer=org.apache.kafka.common.serialization.StringSerializer
value.serializer=org.apache.kafka.common.serialization.StringSerializer

# Topic settings
topic.name=example-topic
topic.partitions=3
topic.replication.factor=1
```

Класс MessageProducer

```
package com.example.kafka.producer;

import org.apache.kafka.clients.producer.KafkaProducer;
import org.apache.kafka.clients.producer.ProducerConfig;
import org.apache.kafka.clients.producer.ProducerRecord;
import org.apache.kafka.clients.producer.RecordMetadata;
import org.apache.kafka.common.serialization.StringSerializer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Properties;
import java.util.concurrent.Future;

public class MessageProducer implements AutoCloseable {
    private static final Logger log = LoggerFactory.getLogger(MessageProducer.class);

    private final KafkaProducer<String, String> producer;
    private final String topic;

    public MessageProducer(String bootstrapServers, String topic) {
        this.topic = topic;

        Properties props = new Properties();
        props.put(ProducerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
        props.put(ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
        props.put(ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG, StringSerializer.class.getName());
        props.put(ProducerConfig.ACKS_CONFIG, "all");
        props.put(ProducerConfig.RETRIES_CONFIG, 3);
        props.put(ProducerConfig.BATCH_SIZE_CONFIG, 16384);
        props.put(ProducerConfig.LINGER_MS_CONFIG, 1);
        props.put(ProducerConfig.BUFFER_MEMORY_CONFIG, 33554432);

        this.producer = new KafkaProducer<>(props);
    }

    public Future<RecordMetadata> send(String key, String value) {
        ProducerRecord<String, String> record = new ProducerRecord<>(topic, key, value);
        log.debug("Sending message to topic {}: key={}, value={}", topic, key, value);
        return producer.send(record, (metadata, exception) -> {
            if (exception != null) {
                log.error("Failed to send message", exception);
            } else {
                log.info("Message sent to partition {} with offset {}",
                        metadata.partition(), metadata.offset());
            }
        });
    }

    public void sendSync(String key, String value) throws Exception {
        send(key, value).get();
    }

    @Override
    public void close() {
        producer.flush();
        producer.close();
        log.info("Producer closed");
    }
}
```

Пример записи Producer

```
2026-01-05 08:00:00.000 [main] INFO  com.example.kafka.Application - Starting application
2026-01-05 08:00:00.123 [main] INFO  com.example.kafka.admin.TopicManager - Creating topic example-topic with 3 partitions
2026-01-05 08:00:00.456 [main] INFO  com.example.kafka.admin.TopicManager - Topic example-topic created successfully
2026-01-05 08:00:00.789 [main] INFO  com.example.kafka.producer.MessageProducer - Producer initialized for topic example-topic
2026-01-05 08:00:01.012 [main] DEBUG com.example.kafka.producer.MessageProducer - Sending message to topic example-topic: key=user-1, value={"name":"Alice","action":"login"}
2026-01-05 08:00:01.234 [kafka-producer-network-thread | producer-1] INFO  com.example.kafka.producer.MessageProducer - Message sent to partition 2 with offset 0
2026-01-05 08:00:01.345 [main] DEBUG com.example.kafka.producer.MessageProducer - Sending message to topic example-topic: key=user-2, value={"name":"Bob","action":"purchase"}
2026-01-05 08:00:01.456 [kafka-producer-network-thread | producer-1] INFO  com.example.kafka.producer.MessageProducer - Message sent to partition 0 with offset 0
2026-01-05 08:00:01.567 [main] DEBUG com.example.kafka.producer.MessageProducer - Sending message to topic example-topic: key=user-3, value={"name":"Carol","action":"logout"}
2026-01-05 08:00:01.678 [kafka-producer-network-thread | producer-1] INFO  com.example.kafka.producer.MessageProducer - Message sent to partition 1 with offset 0
2026-01-05 08:00:01.789 [main] INFO  com.example.kafka.producer.MessageProducer - Producer closed
2026-01-05 08:00:01.890 [main] INFO  com.example.kafka.Application - Application finished
```

Consumer API

В этом разделе мы рассмотрим, как создать потребителя сообщений, который будет читать данные из Kafka. Consumer API позволяет подписываться на топики, получать сообщения и управлять смещениями (offsets).

Конфигурация Consumer

```
# Kafka broker addresses
bootstrap.servers=localhost:9092

# Consumer settings
group.id=example-group
auto.offset.reset=earliest
enable.auto.commit=false
key.deserializer=org.apache.kafka.common.serialization.StringDeserializer
value.deserializer=org.apache.kafka.common.serialization.StringDeserializer
max.poll.records=500
session.timeout.ms=30000
heartbeat.interval.ms=10000

# Topic settings
topic.name=example-topic
```

Класс MessageConsumer

```
package com.example.kafka.consumer;

import org.apache.kafka.clients.consumer.ConsumerConfig;
import org.apache.kafka.clients.consumer.ConsumerRecord;
import org.apache.kafka.clients.consumer.ConsumerRecords;
import org.apache.kafka.clients.consumer.KafkaConsumer;
import org.apache.kafka.common.serialization.StringDeserializer;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.time.Duration;
import java.util.Collections;
import java.util.Properties;
import java.util.function.Consumer;

public class MessageConsumer implements AutoCloseable {
    private static final Logger log = LoggerFactory.getLogger(MessageConsumer.class);

    private final KafkaConsumer<String, String> consumer;
    private final String topic;
    private volatile boolean running = true;

    public MessageConsumer(String bootstrapServers, String groupId, String topic) {
        this.topic = topic;

        Properties props = new Properties();
        props.put(ConsumerConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
        props.put(ConsumerConfig.GROUP_ID_CONFIG, groupId);
        props.put(ConsumerConfig.KEY_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.getName());
        props.put(ConsumerConfig.VALUE_DESERIALIZER_CLASS_CONFIG, StringDeserializer.class.getName());
        props.put(ConsumerConfig.AUTO_OFFSET_RESET_CONFIG, "earliest");
        props.put(ConsumerConfig.ENABLE_AUTO_COMMIT_CONFIG, false);
        props.put(ConsumerConfig.MAX_POLL_RECORDS_CONFIG, 500);

        this.consumer = new KafkaConsumer<>(props);
    }

    public void subscribe() {
        consumer.subscribe(Collections.singletonList(topic));
        log.info("Subscribed to topic {}", topic);
    }

    public void poll(Consumer<ConsumerRecord<String, String>> handler) {
        while (running) {
            ConsumerRecords<String, String> records = consumer.poll(Duration.ofMillis(100));
            for (ConsumerRecord<String, String> record : records) {
                log.debug("Received message: partition={}, offset={}, key={}, value={}",
                        record.partition(), record.offset(), record.key(), record.value());
                handler.accept(record);
            }
            consumer.commitSync();
        }
    }

    public void stop() {
        running = false;
    }

    @Override
    public void close() {
        consumer.close();
        log.info("Consumer closed");
    }
}
```

Пример чтения Consumer

```
2026-01-05 08:00:02.000 [main] INFO  com.example.kafka.Application - Starting consumer
2026-01-05 08:00:02.123 [main] INFO  com.example.kafka.consumer.MessageConsumer - Subscribed to topic example-topic
2026-01-05 08:00:02.234 [main] DEBUG com.example.kafka.consumer.MessageConsumer - Received message: partition=2, offset=0, key=user-1, value={"name":"Alice","action":"login"}
2026-01-05 08:00:02.345 [main] DEBUG com.example.kafka.consumer.MessageConsumer - Received message: partition=0, offset=0, key=user-2, value={"name":"Bob","action":"purchase"}
2026-01-05 08:00:02.456 [main] DEBUG com.example.kafka.consumer.MessageConsumer - Received message: partition=1, offset=0, key=user-3, value={"name":"Carol","action":"logout"}
2026-01-05 08:00:02.567 [main] DEBUG com.example.kafka.consumer.MessageConsumer - Received message: partition=2, offset=1, key=user-4, value={"name":"Dave","action":"login"}
2026-01-05 08:00:02.678 [main] DEBUG com.example.kafka.consumer.MessageConsumer - Received message: partition=0, offset=1, key=user-5, value={"name":"Eve","action":"purchase"}
2026-01-05 08:00:02.789 [main] DEBUG com.example.kafka.consumer.MessageConsumer - Received message: partition=1, offset=1, key=user-6, value={"name":"Frank","action":"logout"}
2026-01-05 08:00:02.890 [main] DEBUG com.example.kafka.consumer.MessageConsumer - Received message: partition=2, offset=2, key=user-7, value={"name":"Grace","action":"login"}
2026-01-05 08:00:02.991 [main] DEBUG com.example.kafka.consumer.MessageConsumer - Received message: partition=0, offset=2, key=user-8, value={"name":"Heidi","action":"purchase"}
2026-01-05 08:00:03.092 [main] DEBUG com.example.kafka.consumer.MessageConsumer - Received message: partition=1, offset=2, key=user-9, value={"name":"Ivan","action":"logout"}
2026-01-05 08:00:03.193 [main] DEBUG com.example.kafka.consumer.MessageConsumer - Received message: partition=2, offset=3, key=user-10, value={"name":"Judy","action":"login"}
2026-01-05 08:00:03.294 [main] INFO  com.example.kafka.consumer.MessageConsumer - Consumer closed
2026-01-05 08:00:03.395 [main] INFO  com.example.kafka.Application - Application finished
```

Admin API

В этом разделе мы рассмотрим, как использовать Admin API для управления топиками, брокерами и конфигурацией кластера. Admin API позволяет создавать, удалять и изменять топики, а также получать информацию о состоянии кластера.

Конфигурация Admin

```
# Kafka broker addresses
bootstrap.servers=localhost:9092

# Admin settings
request.timeout.ms=30000
```

Класс TopicManager

```
package com.example.kafka.admin;

import org.apache.kafka.clients.admin.AdminClient;
import org.apache.kafka.clients.admin.AdminClientConfig;
import org.apache.kafka.clients.admin.NewTopic;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.util.Collections;
import java.util.Properties;
import java.util.Set;
import java.util.concurrent.ExecutionException;

public class TopicManager implements AutoCloseable {
    private static final Logger log = LoggerFactory.getLogger(TopicManager.class);

    private final AdminClient adminClient;

    public TopicManager(String bootstrapServers) {
        Properties props = new Properties();
        props.put(AdminClientConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapServers);
        props.put(AdminClientConfig.REQUEST_TIMEOUT_MS_CONFIG, 30000);

        this.adminClient = AdminClient.create(props);
    }

    public void createTopic(String topicName, int partitions, short replicationFactor)
            throws ExecutionException, InterruptedException {
        NewTopic newTopic = new NewTopic(topicName, partitions, replicationFactor);
        adminClient.createTopics(Collections.singletonList(newTopic)).all().get();
        log.info("Topic {} created with {} partitions and replication factor {}",
                topicName, partitions, replicationFactor);
    }

    public void deleteTopic(String topicName)
            throws ExecutionException, InterruptedException {
        adminClient.deleteTopics(Collections.singletonList(topicName)).all().get();
        log.info("Topic {} deleted", topicName);
    }

    public Set<String> listTopics()
            throws ExecutionException, InterruptedException {
        Set<String> topics = adminClient.listTopics().names().get();
        log.info("Topics: {}", topics);
        return topics;
    }

    @Override
    public void close() {
        adminClient.close();
        log.info("AdminClient closed");
    }
}
```

Пример работы Admin API

```
2026-01-05 08:00:04.000 [main] INFO  com.example.kafka.Application - Starting admin operations
2026-01-05 08:00:04.123 [main] INFO  com.example.kafka.admin.TopicManager - Topics: [example-topic, __consumer_offsets]
2026-01-05 08:00:04.234 [main] INFO  com.example.kafka.admin.TopicManager - Topic test-topic created with 3 partitions and replication factor 1
2026-01-05 08:00:04.345 [main] INFO  com.example.kafka.admin.TopicManager - Topics: [example-topic, test-topic, __consumer_offsets]
2026-01-05 08:00:04.456 [main] INFO  com.example.kafka.admin.TopicManager - Topic test-topic deleted
2026-01-05 08:00:04.567 [main] INFO  com.example.kafka.admin.TopicManager - Topics: [example-topic, __consumer_offsets]
2026-01-05 08:00:04.678 [main] INFO  com.example.kafka.admin.TopicManager - AdminClient closed
2026-01-05 08:00:04.789 [main] INFO  com.example.kafka.Application - Admin operations finished
```

Установка Kafka в режиме KRaft

Автор: Ads.s

Содержание

Обзор
Шаг 1. Добавление сервиса
Шаг 2. Добавление компонентов
Шаг 3. Настройка сервиса
Шаг 4. Установка сервиса
Шаг 5. Проверка результатов
Конфигурационные файлы
Хранение метаданных

Назад к содержанию

Обзор

Сервис Kafka в режиме KRaft устанавливается как обычный сервис при установке кластера ADS и потребует ADCM с применением дополнительных действий:

- добавление дополнительного компонента **Kafka Controller** для создания кворума контроллеров;
- установка параметра **Kafka quorum solution** в значение **[VOTER]** на уровне сервиса кластера сервиса Kafka для создания среды для работы режима KRaft (хранение кластерных метаданных в Kafka).

Новая более подробная статья для установки сервиса Kafka с применением дополнительных действий

- Установка сервиса Kafka в режиме KRaft доступна начиная с версии ADS 3.0.0.2.
- Возможность использования протокола SSH для публикации сервиса в Интернет. Использование протокола предварительно настроено и не рекомендуется для использования в боевой среде. Если используете данную опцию, то обязательно настройте SSL для Kafka и включите клиентскую среду в тех компонентах, которые планируете использовать.
- Для настройки ACL в кластере планируется установка сервиса Kafka в режиме KRaft, установка сервиса Zookeeper не требуется.

Шаг 1. Добавление сервиса

1. В интерфейсе ADCM откройте страницу **Clusters** и кликните по имени кластера ADS. Затем на открывшейся странице кластера перейдите на вкладку **Services** и нажмите **Add services**.

Переход к добавлению сервиса

2. В открывшемся окне выберите сервис **Kafka** и нажмите **Add**.

Выбор сервиса

В результате добавленный сервис отобразится на вкладке **Services**.

Результат успешного добавления сервиса в кластер

Шаг 2. Добавление компонентов

1. На странице кластера откройте вкладку **Mapping**, чтобы задать в распределении сервисных компонентов между хостами кластера.

Переход к распределению сервисных компонентов

Компоненты сервиса Kafka

Компонент	Описание
Kafka Broker	Брокер, отвечающий за хранение партиций топиков, а которые подключаются клиенты для записи или чтения сообщений
Kafka Controller	Брокер, который отвечает за хранение топика метаданных кластера, участник кворума контроллеров

2. Назначьте хост каждому компоненту сервиса Kafka – нажмите **Add hosts** и выберите нужный хост из открывшегося окна.

Выбор хоста для компонента

ВНИМАНИЕ

Для сервиса Kafka в режиме KRaft Kafka Controller является обязательным компонентом, даже если он не задается вручную цветом. Рекомендуемое количество контроллеров Kafka Controller на кластер - 3.

3. После завершения распределения компонентов нажмите **Save**, чтобы сохранить изменения.

Сохранение распределения компонентов

Шаг 3. Настройка сервиса

1. Откройте вкладку **Services** на странице кластера и кликните по имени сервиса **Kafka** и столбцу **Name**.

Переход к настройке сервиса

2. В открывшемся окне **Primary configuration**

- Kafka quorum solution в значение **[VOTER]**
- При необходимости настройте остальные параметры сервиса – описание параметров приведено в разделе **Kafka** – блок Конфигурационные параметры ADS.

Настройка сервиса Kafka

После указания всех необходимых параметров нажмите **Save**.

3. Перейдите к настройке компонента Kafka Controller. Для этого во вкладке **Primary configuration** перейдите на вкладку **Components** и выберите компонент **Kafka Controller**.

Переход к настройке компонента Kafka Controller

4. В открывшемся окне настройте параметры Kafka Controller. **Allow advanced** и при необходимости измените значения на нужные, установленные для конфигурационного параметра компонента Kafka Controller – **controller.listener.names**. Значение должно совпадать с listener.security.protocol.map, заданным в параметре **Kafka** блока Конфигурационные параметры ADS.

Настройка компонента Kafka Controller

После указания всех необходимых параметров нажмите **Save**.

Шаг 4. Установка сервиса

1. На вкладке **Services** в правом верхнем углу кластера нажмите на кнопку **Actions** и в списке действий выберите **Install**.

Переход к установке сервиса

2. Подтвердите действие в открывшемся окне.

Подтверждение действия

3. Дождитесь завершения установки и убедитесь, что статус сервиса изменился с **Installed** на **Installed**.

Процесс установки завершен

Чтобы посмотреть процесс выполнения и изучить журнал установки и возможность ошибок в случае их возникновения, выберите пункт **Jobs** в левом навигационном меню и кликните по имени задания **Install** **Jobs**.

Журнал задач Install

Шаг 5. Проверка результатов

Конфигурационные файлы

После установки Kafka в режиме KRaft через интерфейс ADCM автоматически устанавливаются все необходимые параметры для работы в конфигурационных файлах и скриптах запуска. Для просмотра файлов конфигурации можно воспользоваться Web-Shell/SSH. Также предварительно при помощи ADCM можно конфигурировать параметры Kafka, которые попадут в конфигурационные файлы в соответствующие разделы.

Конфигурационный файл брокера Kafka server.properties

```
node.id=1001
controller.quorum.voters=1001@ads-vm-test-1.ru-central1.internal:9093
listeners=SASL_PLAINTEXT://:9092,CONTROLLER://:9093
inter.broker.listener.name=SASL_PLAINTEXT
advertised.listeners=SASL_PLAINTEXT://ads-vm-test-1.ru-central1.internal:9092
log.dirs=/kafka/data
num.partitions=1
num.recovery.threads.per.data.dir=1
offsets.topic.replication.factor=1
transaction.state.log.replication.factor=1
transaction.state.log.min.isr=1
log.retention.hours=168
log.segment.bytes=1073741824
log.retention.check.interval.ms=300000
zookeeper.connect=localhost:2181
zookeeper.connection.timeout.ms=18000
group.initial.rebalance.delay.ms=0
security.inter.broker.protocol=SASL_PLAINTEXT
sasl.mechanism.inter.broker.protocol=PLAIN
sasl.enabled.mechanisms=PLAIN
authorizer.class.name=kafka.security.authorizer.AclAuthorizer
super.users=User:admin
listener.security.protocol.map=CONTROLLER:SASL_PLAINTEXT,SASL_PLAINTEXT:SASL_PLAINTEXT
```

Конфигурационный файл Kafka Controller kafka-controller.properties

```
process.roles=controller
node.id=1003
controller.quorum.voters=1001@ads-vm-test-1.ru-central1.internal:9093
listeners=CONTROLLER://:9093
log.dirs=/kafka/data
controller.quorum.voters=1001@ads-vm-test-1.ru-central1.internal:9093
listener.security.protocol.map=CONTROLLER:PLAINTEXT
security.inter.broker.protocol=PLAINTEXT
```

Файл для хранения переменных окружения kafka-controller-env.sh

```
export LOG_DIR=/var/log/kafka-controller
export PID_DIR=/var/run/kafka-controller
```

Хранение метаданных

Для хранения метаданных сервиса в режиме KRaft ADCM при первом старте на базе Kafka Kafka создал от имени пользователя кворум контроллеров, который хранит метаданные кластера. Информация о кворуме хранится в файле /kafka/data/__cluster_metadata-0/quorum-state, содержащем данные:

- clusterId: ClusterId – идентификатор набора данных (uuid) для хранения метаданных (кворум контроллеров Kafka), которые используются для начальной загрузки кластера;
- __cluster_metadata-0 – директория, содержащая файлы, в которых сохраняются метаданные кластера Kafka;
- snapshot.checkpoint – файл, содержащий информацию о текущем снимке, версии и идентификаторе кластера.

Миграция Kafka на KRaft

Ольга Семина

AD.S

Содержание

Назад к содержанию

Обзор

Подготовка к миграции

- Определение состояния кластера
- Установка контроллеров Kafka

Включение режима миграции

Проверка миграции данных

Выключение режима миграции и установка режима KRaft для Kafka

Обзор

По умолчанию в ADCM в кластере ADS для сервиса Kafka управление метаданными осуществляется при помощи сервиса ZooKeeper, который обеспечивает работу кворума, основанную на алгоритме консенсуса ZAB.

Также сервис Kafka может быть установлен сразу в режиме KRaft для использования алгоритма консенсуса Raft.

Для кластеров, в которых сервис Kafka уже используется вместе с ZooKeeper, может быть выполнена миграция в режим KRaft.

ВНИМАНИЕ

- Миграция кластера Kafka в режим KRaft доступна начиная с версии ADS 3.7.2.1.b1.
- Возможность использования протокола KRaft для Kafka предоставляется в режиме технической предварительной версии и не рекомендуется для использования в продуктовой среде. Arenadata рекомендует вам изучить эту функцию в непроизводственных средах и при необходимости оставить отзыв о своем опыте.
- Возможность обратной миграции из KRaft в ZooKeeper отсутствует.

Процесс миграции является полуавтоматическим и включает в себя следующие этапы:

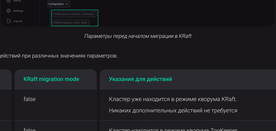
1. Подготовка к миграции
2. Включение режима миграции
3. Проверка миграции данных
4. Выключение режима миграции и установка режима KRaft для Kafka

Включение и выключение режима миграции выполняется при помощи действий сервиса Kafka в интерфейсе ADCM, может запуска каждого их них определяется пользователем.

Подготовка к миграции

Определение состояния кластера

Перед включением миграции на странице конфигурирования сервиса Kafka проверьте состояние параметров ***Kafka quorum solution*** и ***KRaft migration mode***.



Параметры перед началом миграции в KRaft

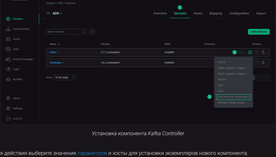
Ниже описаны указания для действий при различных значениях параметров.

Kafka quorum solution	KRaft migration mode	Указания для действий
KRaft	false	Кластер уже находится в режиме кворума KRaft. Никаких дополнительных действий не требуется
ZooKeeper	false	Кластер находится в режиме кворума ZooKeeper. Для кластера допускается включение режима миграции в KRaft. Перед включением режима миграции установите компонент Kafka Controller
ZooKeeper	true	Режим миграции уже запущен. Выполните проверку миграции данных и переходите к выключению режима миграции

Установка контроллеров Kafka

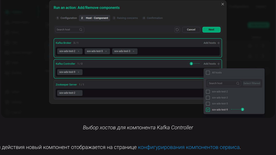
Для перехода на режим KRaft в кластере для сервиса Kafka должен быть установлен новый компонент — Kafka Controller. Рекомендуемое количество экземпляров компонента ***Kafka Controller*** на кластер — 3.

Для добавления экземпляров нового компонента сервиса Kafka в интерфейсе ADCM запустите действие ***Add/Remove components***, нажав на иконку в столбце ***Actions***.



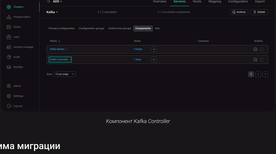
Установка компонента Kafka Controller

Далее в форме подтверждения действия выберите значения параметров и хосты для установки экземпляров нового компонента.



Выбор хостов для компонента Kafka Controller

После успешного выполнения действия новый компонент отображается на странице конфигурирования компонентов сервиса.



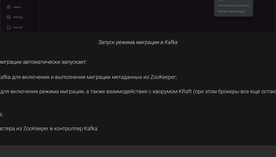
Компонент Kafka Controller

Включение режима миграции

Для запуска режима миграции должны быть соблюдены следующие условия:

- Состояние параметров сервиса Kafka:
  - ***Kafka quorum solution*** — в значении `ZooKeeper`;
  - ***KRaft migration mode*** — в значении `false`.
- В сервисе Kafka установлен компонент ***Kafka Controller***, как описано выше.

Для запуска режима миграции в интерфейсе ADCM запустите действие ***KRaft migration stage 1*** для сервиса Kafka, нажав на иконку в столбце ***Actions***.



Запуск режима миграции в Kafka

Процесс включения режима миграции автоматически запускает:

- настройку контроллеров Kafka для включения и выполнения миграции метаданных из ZooKeeper;
- настройку брокеров Kafka для включения режима миграции, а также взаимодействие с кворумом KRaft (при этом брокеры все еще остаются настроены для работы с ZooKeeper);
- перезапуск брокеров Kafka;
- миграцию метаданных кластера из ZooKeeper в контроллер Kafka.

ВНИМАНИЕ

- После успешного выполнения включения режима миграции контроллеры Kafka полностью функциональны и извлекают метаданные из ZooKeeper, но брокеры Kafka по-прежнему используют ZooKeeper в качестве кворума.
- Переход к выключению режима миграции допускается выполнять только после проверки миграции данных, иначе может возникнуть поломка кластера.

Ниже приведено состояние параметров после успешного включения режима миграции:

- ***Kafka quorum solution*** — в значении `ZooKeeper`;
- ***KRaft migration mode*** — в значении `true`.

Проверка миграции данных

В логе контроллеров Kafka в течение миграции метаданных появляются соответствующие сообщения.

Все сообщения можно просмотреть, обратившись к файлу /var/log/kafka-controller/server.log на хосте, где размещен компонент ***Kafka Controller***.

Миграция данных является законченной, когда в логе появляется запись:

```
Completed migration of metadata from Zookeeper to KRaft.
```

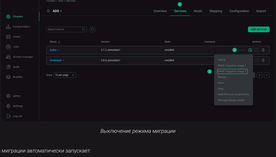
Только в случае появления данного сообщения допускается переходить к выключению режима миграции.

Выключение режима миграции и установка режима KRaft для Kafka

Для выключения режима миграции должно быть соблюдены следующие условия:

- Состояние параметров сервиса Kafka:
  - ***Kafka quorum solution*** — в значении `ZooKeeper`;
  - ***KRaft migration mode*** — в значении `true`.
- *Лог контроллеров Kafka содержит запись о завершении миграции данных из ZooKeeper в KRaft, как описано выше.*

Для выключения режима миграции в интерфейсе ADCM запустите действие ***KRaft migration stage 2*** для сервиса Kafka, нажав на иконку в столбце ***Actions***.



Выключение режима миграции

Процесс выключения режима миграции автоматически запускает:

- настройку контроллеров Kafka для удаления всех свойств, связанных с ZooKeeper, и завершения миграции;
- настройку брокеров Kafka для указания напрямую на кворум KRaft вместо кворума ZooKeeper, удаления всех свойств, связанных с ZooKeeper, и завершения миграции;
- перезапуск брокеров и контроллеров Kafka.

Ниже приведено состояние параметров после успешного выключения режима миграции:

- ***Kafka quorum solution*** — в значении `KRaft`;
- ***KRaft migration mode*** — в значении `false`.

Настройка и использование Kafka Tiered Storage

Kirill Sukhov

Содержание

Обзор

Hadoop Distributed File System (HDFS)

- Предварительные требования
- Шаг 1. Импорт данных ADH
- Шаг 2. Настройка и включение Tiered Storage
- Шаг 3. Проверка результатов

Simple Storage Service (S3)

- Предварительные требования
- Шаг 1. Настройка и включение Tiered Storage
- Шаг 2. Проверка результатов

Использование Tiered Storage

Назад к содержанию

Обзор

Open Tiered Storage в ADS может быть включен в конфигурациях параметром сервиса Kafka (определяемых через HDFS или S3).

Также настройки описаны шаг за включения Tiered Storage

Hadoop Distributed File System (HDFS)

Для данного примера в качестве целевым хранилищем для сегментов HDFS кластер Managed Hadoop (MDH).

Предварительные требования

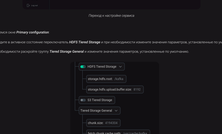
Для включения Tiered Storage в Kafka используются следующие окружение:

- Кластер ADS развернут согласно документации (Yandex Cloud или Terraform). Минимальная версия ADS — 1.0.1.231.
- Сервисы Kafka и Zookeeper установлены в кластере ADS.
- Кластер ADS развернут согласно документации (Yandex Cloud или Terraform). Минимальная версия ADH — 3.3.6.23.
- Сервисы HDFS, Core configuration и Zookeeper установлены в кластере ADH.
- В HDFS необходимо создать директорию, путь и название которой совпадает с указываемым в качестве конфигурационного параметра **storage.hdfs.root**, и настроить права доступа:

```
$ sudo -u hdfs hdfs dfs -mkdir /kafka
$ sudo -u hdfs hdfs dfs -chown kafka:hadoop /kafka
```

Шаг 1. Импорт данных ADH

В интерфейсе ADCM откройте страницу **Clusters** и кликните по имени кластера ADS. Затем на открывшейся странице кластера перейдите на вкладку **Import**, выберите **Cluster configuration** напротив названия кластера ADH и нажмите **Import**.



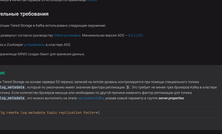
Импорт данных ADH

Шаг 2. Настройка и включение Tiered Storage

1. Откройте вкладку **Services** на странице кластера и кликните по имени сервиса **Kafka** в столбце **Name**.

Переход к настройке сервиса

2. В открывшейся вкладке **Primary configuration**:
   - Переведите в активное состояние переключатель **HDFS Tiered Storage** и при необходимости измените значения параметров, установленные по умолчанию.
   - При необходимости разверните группу **Tiered Storage (General)** и измените значения параметров, установленных по умолчанию.



Настройка Tiered Storage

Описание параметров приведено в разделе **Kafka** (Yandex Cloud) документации (параметры S3).

3. Нажмите **Save** и подтвердите кнопкой **Kafka**. Как только для сервиса **Restart**, кликом на иконку ↻ и кнопкой **Actions**.
4. Дождитесь завершения перезагрузки сервиса. Проконтролируйте его работу в столбце **Status** на основном экране на странице **Jobs**.

Шаг 3. Проверка результатов

- При включенном слове Tiered Storage можно настройки ADCM автоматически устанавливается все необходимые параметры для работы с хранилищем.
- В конфигурационном файле брокера Kafka (в директории /etc/kafka/conf) устанавливается через systemd-сервис Remote Manager. StorageBackend (определяющего) и дистанционно для выбранного типа хранилища и включения кластер.
Также примеры правил конфигурации Tiered Storage ADCM файла конфигурации

Конфигурационный файл брокера Kafka server.properties

```
# Managed by ADCM

node.id=1
reserved.broker.max.id=65000
auto.create.topics.enable=false
listeners=SASL_PLAINTEXT://:9092
log.dirs=/kafka/logs
default.replication.factor=1
num.partitions=1
delete.topic.enable=true
log.retention.hours=168
log.roll.hours=168
queued.max.requests=500
num.io.threads=8
num.io.threads=8
auto.leader.rebalance.enable=true
unclean.leader.election.enable=true
offsets.topic.replication.factor=1
transaction.state.log.replication.factor=1
transaction.state.log.min.isr=1

zookeeper.connect=vmz-kafka-1.ru-central1.internal:2181/arenadata1?clusterId=1
zookeeper.connection.timeout.ms=30000
zookeeper.session.timeout.ms=30000
zookeeper.sync.time.ms=2000
zookeeper.set.acl=false

log.flush.scheduler.interval.ms=2000
log.cleanup.policy=delete
log.flush.interval.ms=2000
log.cleaner.min.compaction.lag.ms=0
log.cleaner.delete.retention.ms=86400000

security.inter.broker.protocol=PLAINTEXT

#
#
#
#
#
#
#remote.log.metadata.manager.listener.name=PLAINTEXT
#
#remote.log.storage.manager.class.name=io.aiven.kafka.tieredstorage.RemoteStorageManager
#remote.log.storage.manager.class.path=/usr/lib/kafka/libs/tiered-storage/*
#remote.log.storage.system.enable=true
#rsm.config.storage.backend.class=
#rsm.config.fetch.chunk.cache.class=io.aiven.kafka.tieredstorage.fetch.cache.DiskChunkCache
#rsm.config.fetch.chunk.cache.path=/var/tmp/kafka/tscache
#rsm.config.fetch.chunk.cache.retention.ms=1209600000
#rsm.config.fetch.chunk.cache.size=10737418240
#rsm.config.storage.aws.access.key.id=[ACCESS_KEY_ID]
#rsm.config.storage.backend.class=io.aiven.kafka.tieredstorage.storage.hdfs.HdfsStorage
#rsm.config.storage.hdfs.core.path=/usr/lib/kafka/config/core-site.xml
#rsm.config.storage.hdfs.hdfs.path=/usr/lib/kafka/config/hdfs-site.xml
#rsm.config.storage.hdfs.root=/kafka
#rsm.config.storage.hdfs.upload.buffer.size=8192
```

3. В конфигурационном файле переменных окружения /etc/kafka/conf/kafka-env.sh установите значение для параметра **KAFKA_ZOOKEEPER_OPTS**, чтобы путь соответствовать файлу конфигурации ACH:

```
export KAFKA_ZOOKEEPER_OPTS="-Djute.maxbuffer=0xfffff"
```

- При включении слова Tiered Storage в папку /usr/lib/kafka/config/ копируются конфигурационные файлы ACH:
  - rsm-advanced
  - rsm-secret

Simple Storage Service (S3)

Для данного примера в качестве сервиса S3 используется хранилище MINIO.

Предварительные требования

Для включения слова Tiered Storage в Kafka используются следующие окружение:

- Кластер ADS развернут согласно документации (Yandex Cloud или Terraform). Минимальная версия ADS — 1.0.2.31.
- Сервисы Kafka и Zookeeper установлены в кластере ADS.
- В облачном хранилище MINIO создан бакет для хранения данных.

> **ПРИМЕЧАНИЕ**
>
> В реализации Tiered Storage из основе сервиса S3 можно запускать на кластере данных конфигурируется при помощи специального плагина **_CONFIG_TAG_METADATA**, которые по умолчанию имеют значение Kafka Tiered Storage для Kafka в кластере для работы плагина. Если значение Kafka Tiered Storage для Kafka в кластере для работы плагина, укажите имя параметра **_CONFIG_TAG_METADATA**, указав новый параметр в группе **server.properties**.

```
rlm.config.remote.log.metadata.topic.replication.factor=1
```

Шаг 1. Настройка и включение Tiered Storage

1. Откройте вкладку **Services** на странице кластера и кликните по имени сервиса **Kafka** в столбце **Name**.

Переход к настройке сервиса

2. В открывшейся вкладке **Primary configuration**:
   - Для сервиса S3, с которым служба не бакет задается и в формате TCP/S, а в виде пути (директива, для MINIO ссылка на бакет выглядит так: http://<Имя/Имя>/<Имя/Имя>/), имя бакета и директория, куда будет записываться данные. Для этого необходимо указать соответствующий ему ссылку в бакету

```
rsm.config.storage.s3.path.style.access.enable=true
```

Для этого раскройте группу **server.properties** и включите поле **Add key:value**, выберите **All property** и введите название параметра и его значение.

- Переведите в активное состояние переключатель **S3 Tiered Storage** и введите значения параметров для подключения к хранилищу S3.

> **ПРИМЕЧАНИЕ**
>
> Значение параметра **storage.s3.region** не может быть пустым. Если он задано S3 не предоставлен такой параметр, установите для него любое значение, например, **none**.

- При необходимости разверните группу **Tiered Storage (General)** и измените значения параметров, установленных по умолчанию.

Настройка Tiered Storage

Описание параметров приведено в разделе **Kafka** (Yandex Cloud) документации (параметры S3).

3. Нажмите **Save** и подтвердите кнопкой **Kafka**. Как только для сервиса **Restart**, кликом на иконку ↻ и кнопкой **Actions**.
4. Дождитесь завершения перезагрузки сервиса. Проконтролируйте его работу в столбце **Status** на основном экране на странице **Jobs**.

Шаг 2. Проверка результатов

При включении слова Tiered Storage можно настройки ADCM автоматически устанавливается все необходимые параметры для работы с хранилищем S3, определяемой (определяющего) дистанционно для выбранного типа хранилища и включения кластер.
Также примеры правил конфигурации Tiered Storage ADCM файла конфигурации

Конфигурационный файл брокера Kafka server.properties

```
# Managed by ADCM

node.id=1
reserved.broker.max.id=65000
auto.create.topics.enable=false
listeners=SASL_PLAINTEXT://:9092
log.dirs=/kafka/logs
default.replication.factor=1
num.partitions=1
delete.topic.enable=true
log.retention.hours=168
log.roll.hours=168
queued.max.requests=500
num.io.threads=8
num.io.threads=8
auto.leader.rebalance.enable=true
unclean.leader.election.enable=true
offsets.topic.replication.factor=1
transaction.state.log.replication.factor=1
transaction.state.log.min.isr=1

zookeeper.connect=vmz-adh-test-2.ru-central1.internal:2181,vmz-adh-test-3.ru-central1.internal:2181,vmz-adh-test-4.ru-central1.internal:2181/arenadata1?clusterId=1
zookeeper.connection.timeout.ms=30000
zookeeper.session.timeout.ms=30000
zookeeper.sync.time.ms=2000
zookeeper.set.acl=false

log.flush.scheduler.interval.ms=2000
log.cleanup.policy=delete
log.flush.interval.ms=2000
log.cleaner.min.compaction.lag.ms=0
log.cleaner.delete.retention.ms=86400000

#remote.log.metadata.manager.listener.name=PLAINTEXT
#
#remote.log.storage.manager.class.name=io.aiven.kafka.tieredstorage.RemoteStorageManager
#remote.log.storage.manager.class.path=/usr/lib/kafka/libs/tiered-storage/*
#remote.log.storage.system.enable=true
#rsm.config.storage.bucket.name=
#rsm.config.storage.s3.endpoint.url=
#rsm.config.fetch.chunk.cache.class=io.aiven.kafka.tieredstorage.fetch.cache.DiskChunkCache
#rsm.config.fetch.chunk.cache.path=/var/tmp/kafka/tscache
#rsm.config.fetch.chunk.cache.retention.ms=1209600000
#rsm.config.fetch.chunk.cache.size=10737418240
#rsm.config.storage.aws.access.key.id=[ACCESS_KEY_ID]
#rsm.config.storage.aws.secret.access.key=[SECRET_ACCESS_KEY]
#rsm.config.storage.s3.region=
#rsm.config.storage.s3.path.style.access.enable=true
#rsm.config.storage.s3.multipart.upload.part.size=26214400

security.inter.broker.protocol=PLAINTEXT

#remote.log.metadata.manager.listener.name=PLAINTEXT
#
#remote.log.storage.manager.class.name=io.aiven.kafka.tieredstorage.RemoteStorageManager
#remote.log.storage.manager.class.path=/usr/lib/kafka/libs/tiered-storage/*
#remote.log.storage.system.enable=true
#rsm.config.fetch.chunk.cache.class=io.aiven.kafka.tieredstorage.fetch.cache.DiskChunkCache
#rsm.config.fetch.chunk.cache.path=/var/tmp/kafka/tscache
#rsm.config.fetch.chunk.cache.retention.ms=1209600000
#rsm.config.fetch.chunk.cache.size=10737418240
#rsm.config.storage.backend.class=io.aiven.kafka.tieredstorage.storage.s3.S3Storage
#rsm.config.storage.s3.endpoint.url=http://[hostname:port]
#rsm.config.storage.s3.bucket.name=[bucket_name]
#rsm.config.storage.aws.access.key.id=[access_key]
#rsm.config.storage.aws.secret.access.key=[secret_key]
#rsm.config.storage.s3.region=[region]
```

Использование Tiered Storage

После успешных настроек сегменты которого должны быть перенесены в хранилище, необходимо для данного топика создать Tiered Storage (как только параметры remote.storage.enable).

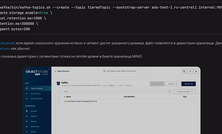
Если для топика не задано параметр log.retention.ms, время локального хранения данных будет соответствовать времени указанному для retention.

Ниже приведен пример создания топика с указанием параметров, а также указанием retention.ms=10000 (в миллисекундах):

```
$ /usr/lib/kafka/bin/kafka-topics.sh --create --topic tieredTopic --bootstrap-server adh-test-1.ru-central1.internal:9092 \
--config remote.storage.enable=true \
--config local.retention.ms=1000 \
--config retention.ms=600000 \
--config segment.bytes=100
```

Перед началом загрузки данных всем журнал локального хранения и после от сегмент достиг указанного размера, файл помещается в директории хранилище. Данные сообщения могут быть прочитаны как обычно.

Также для каждого названия сегмента создаются три файла с общим именем и идентификатором сегмента журнала.



Директория с сообщениями топика на каталог диске

Созданная в хранилище директория tieredTopic-0Hpvm7aTaGqrSiYI7VJ-mAAA имеет имя, состоящее из названия топика tieredTopic, и случайно следует через и идентификатора топика. Эта директория в свою очередь содержит пять следующих типов файлов:

- 00000000000000000000.index — файл, содержащий индексы смещений для сообщений журнала и указывает сегментов журнала.
- 00000000000000000000.log — файл, являющийся сегментом журнала, содержит записи Kafka.
- 00000000000000000000.snapshot — файл producer-id.snapshot — файл, содержащий моментальных снимков состояния журнала и индексов.
- 00000000000000000000.timeindex — файл, содержащий смещение и идентификатора сегмента журнала.

Содержание

Назад к содержанию

Обзор репликации на базе Kafka Connect
Настройка
Проверка репликации топиков

Обзор репликации на базе Kafka Connect

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

Kafka Connect доступен к установке в ADS начиная с версии 1.7.1.

При архитектуре репликации **Active/Active** кластеры получают данные непосредственно от источников данных и реплицированные данные, вводимые из удаленного кластера.

При архитектуре репликации **Active/Standby** целевой кластер находится в пассивном режиме (не имеет подключенных к нему потребителей и производителей) и получает только реплицированные данные.

На рисунке показана последовательность механизма репликации.

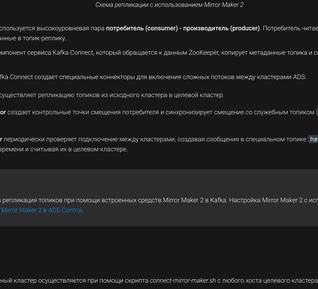


Схема репликации с использованием Mirror Maker 2

Для перемещения данных используется высокоуровневая пара **потребитель (consumer) - производитель (producer)**. Потребитель читает и обрабатывает исходный топик, производитель помещает данные в топик-реплику.

Replicator – специальный компонент сервиса Kafka Connect, который обращается к данным ZooKeeper, копирует метаданные топика и создает такой же топик в целевом кластере.

MirrorMaker 2.0 в составе Kafka Connect создает специальные коннекторы для включения сложных потоков между кластерами ADS:

- **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.

ПРИМЕЧАНИЕ

В данной статье описана репликация топиков при помощи встроенных средств Mirror Maker 2 в Kafka. Настройка Mirror Maker 2 с использованием ADS Control описана в статье Mirror Maker 2 в ADS Control.

Настройка

Репликация топика на удаленный кластер осуществляется при помощи скрипта *connect-mirror-maker.sh* с любого хоста целевого кластера ADS.

Скрипт connect-mirror-maker.sh создает и настраивает коннекторы на основе предоставленного файла свойств mm2.properties.

Для настройки конфигурационного файла mm2.properties необходимо:

1. Ввести команду для открытия (создания) файла:

```
$ sudo vim /usr/lib/kafka/bin/mm2.properties
```

2. Заполнить файл данными:

```
clusters = <source_cluster_alias>, <target_cluster_alias>
<target_cluster_alias>.bootstrap.servers = hostname1:9092,hostname2:9092,hostname3:9092
<source_cluster_alias>.bootstrap.servers = hostname4:9092,hostname5:9092,hostname6:9092

topics
topics =<test_topic_name>
groups = *

<source_cluster_alias>->target_cluster_alias> enabled=true

replication.factor=3

checkpoints.topic.replication.factor=3
heartbeats.topic.replication.factor=3
offset-syncs topic.replication.factor=3

offset.storage.replication.factor=3
status.storage.replication.factor=3
config.storage.replication.factor=3
```

Описание содержания mm2.properties

Параметры	Описание
clusters = <source_cluster_alias>, <target_cluster_alias>	Имена для каждого кластера, которые будут использоваться в репликации, где (здесь и ниже): • `<source_cluster_alias>` – кластер, являющийся источником данных; • `<target_cluster_alias>` – кластер, куда должны быть перемещены данные (целевой).
<source_cluster_alias> bootstrap.servers = hostname1:9092,hostname2:9092,hostname3:9092 <target_cluster_alias> bootstrap.servers = hostname4:9092,hostname5:9092,hostname6:9092	Информация о соединении для каждого кластера – пары `хост:порт 9092`, разделенные запятыми
<source_cluster_alias> -> <target_cluster_alias> enabled = true	Включение и обозначение направления репликации, в примере включена репликация из кластера `<source_cluster_alias>` в кластер `<target_cluster_alias>`
topics topics = <test_topic_name> groups = *	Название топиков или групп, предназначенных для репликации, где `<test_topic_name>` – имя одного или нескольких топиков, перечисленных через запятую
replication.factor=1	Коэффициент репликации вновь созданных топиков-реплик
heartbeats.topic.replication.factor=3 checkpoints.topic.replication.factor=3 offset-syncs topic.replication.factor=3	Фактор репликации для служебных топиков MM2: • `heartbeats` – топик проверки соединения в каждом исходном кластере, который реплицируется для демонстрации подключения через коннекторы. Сообщения в этом топике содержат информацию об исходном кластере, целевом кластере и отметке времени создания heartbeat. • `source.checkpoints.internal` – топик в целевом кластере для записи смещений для каждой группы потребителей в исходном кластере. • `mm2-offset-syncs.<target_cluster_alias>.internal` – заполняется MirrorSourceConnector, а затем используется MirrorCheckpointConnector для изменения смещений группы потребителей.
offset.storage.replication.factor=3 status.storage.replication.factor=3 config.storage.replication.factor=3	Фактор репликации для создания внутренних топиков кластеров: • `mm2-configs.<target_cluster_alias>.internal` • `mm2-offsets.<target_cluster_alias>.internal` • `mm2-status.<target_cluster_alias>.internal`
replication.policy.separator = _	Если значение оставить пустым, топик-реплика будет иметь то же имя, что и исходный топик
sync.topic.acls.enabled = false	Включение мониторинга исходного кластера на предмет изменений ACL
emit.heartbeats.interval.seconds = 5	Интервал для сообщений heartbeat

Проверка репликации топиков

Для того чтобы запустить скрипт *connect-mirror-maker.sh* и создать коннекторы, необходимо ввести на любом брокере целевого кластера команду:

```
$ /usr/lib/kafka/bin/connect-mirror-maker.sh mm2.properties
```

Далее запускаются коннекторы, которые циклично опрашивают исходный топик для считывания сообщений. Эту терминальную сессию можно не завершать, тогда сообщения можно будет считывать в реальном времени.

В процессе работы коннекторов создаются внутренние топики кластера:

- на стороне исходного кластера:
  - mm2-configs.<target_cluster_alias>.internal
  - mm2-offset-syncs.<target_cluster_alias>.internal
  - mm2-offsets.<target_cluster_alias>.internal
  - mm2-status.<target_cluster_alias>.internal
    где `<target_cluster_alias>` соответствует обозначению целевого кластера, введенному при создании *mm2.properties*.
- на стороне целевого кластера:
  - mm2-configs.<source_cluster_alias>.internal
  - mm2-offset-syncs.<source_cluster_alias>.internal
  - mm2-offsets.<source_cluster_alias>.internal
  - mm2-status.<source_cluster_alias>.internal
    где `<source_cluster_alias>` соответствует обозначению исходного кластера, введенному при создании *mm2.properties*.

Для того чтобы проверить наличие служебных топиков, необходимо ввести на *любом брокере* кластера (в новой терминальной сессии) команду для вывода списка топиков:

```
$ /usr/lib/kafka/bin/kafka-topics.sh --list --bootstrap-server hostname:9092
```

В списке топиков должны присутствовать нужные топики.

Создание реплицируемого исходного топика и запись в него сообщений происходит на стороне исходного кластера.

Для создания исходного топика необходимо ввести команду:

```
$ /usr/lib/kafka/bin/kafka-topics.sh --create --topic <test_topic_name> --bootstrap-server hostname:9092
```

где `<test_topic_name>` – имя исходного топика, введенное при создании *mm2.properties*.

Для записи сообщений в исходный топик необходимо ввести команду:

```
$ /usr/lib/kafka/bin/kafka-console-producer.sh --topic <test_topic_name> --bootstrap-server hostname:9092
```

После записи сообщений в исходный топик в списке топиков на целевом кластере появляется топик-реплика с названием в формате `<source_cluster_alias>.<test_topic_name>`, *где:*

- `<source_cluster_alias>` – соответствующее обозначение исходного кластера, введенное при создании *mm2.properties*.
- `<test_topic_name>` – имя исходного топика, введенное при создании *mm2.properties*.

Для того чтобы считать сообщения из топика-реплики на целевом кластере, необходимо ввести команду:

```
$ /usr/lib/kafka/bin/kafka-console-consumer.sh --topic <source_cluster_alias>.<test_topic_name> --from-beginning --bootstrap-server hostname:9092
```

Считываемые сообщения в целевом кластере полностью повторяют записанные в исходном кластере.

Пример использования Kafka REST Proxy

Ольга Семме

Содержание

Назад к содержанию

Запись сообщений в топик

Чтение сообщений из топика

Получение данных о топиках

- Список топиков
- Конфигурационные параметры топиков
- Партиции

В статье описан пример использования сервиса Kafka REST Proxy – работа с топиками Kafka по интерфейсу RESTful при помощи инструмента командной строки cURL.

Запись сообщений в топик

Для записи в топик используется форма запроса POST /topics/{string:topic_name}.

Запрос для записи пары ключ/значение `{"name":"Helen"}` в JSON-формате в топик `my_topic` выглядит следующим образом:

```
$ curl -X POST -H "Content-Type: application/vnd.kafka.json.v2+json" \
      --data '{"records":[{"value":{"name":"Helen"}}]}' http://hostname:8082/topics/my_topic
```

где:

- `X POST` – определяет метод HTTP-запроса, который используется для отправки данных на сервер.
- `H "Content-Type: application/vnd.kafka.json.v2+json"` – определяет пользовательский заголовок, описывающий тип контента тела запроса.
- `data '{"records":[{"value":{"name":"Helen"}}]}'` – данные, описывающие тело запроса. Тело запроса содержит информацию, необходимую для создания или изменения объекта.
- `http://hostname:8082/topics/my_topic` – конечная точка.
- `hostname` – имя хоста, где установлен сервис Kafka REST Proxy.

ПРИМЕЧАНИЕ

Запросы, приведенные далее, имеют составные части, аналогичные запросу для записи сообщений в топик.

Результат запроса при успешной записи сообщений:

```
{"offsets":[{"partition":0,"offset":0,"error_code":null,"error":null}],"key_schema_id":null,"value_schema_id":null}
```

ВНИМАНИЕ

При записи сообщений в топик необходимо учитывать следующее:

- Для автоматического создания топика необходимо включить параметр ***auto.create.topics.enable*** в группе ***server.properties*** при конфигурировании сервиса Kafka.
- Наименование топика может включать буквенно-цифровые символы (в том числе в кодировке ASCII), а также символы `.`, `_` и `-`. При использовании других символов будет сформирована ошибка.

Чтение сообщений из топика

Для чтения сообщения из топика необходимо выполнить:

1. Создать потребителя (consumer). Для этого используется форма запроса POST /consumers/{string:group_name}.
   Запрос на создание потребителя `my_consumer` в группе потребителей `my_group` для считывания данных JSON-формата с начала топика выглядит следующим образом:

```
$ 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
```

   Результат запроса при успешном создании потребителя:

```
{"instance_id":"my_consumer","base_uri":"http://hostname:8082/consumers/my_group/instances/my_consumer"}
```

2. Подписаться на топик. Для этого используется форма запроса POST /consumers/{string:group_name}/instances/{string:instance}/subscription.
   Для подписки на топик необходимо использовать базовый URL-адрес `base_uri`, полученный в результате запроса на создание потребителя.

   Запрос на подписку на топик `my_topic` для потребителя `my_consumer` из группы `my_group` выглядит следующим образом:

```
$ curl -X POST -H "Content-Type: application/vnd.kafka.v2+json" --data '{"topics":["my_topic"]}' \
 http://hostname:8082/consumers/my_group/instances/my_consumer/subscription
```

3. Считать сообщения из топика. Для этого используется форма запроса GET /consumers/{string:group_name}/instances/{string:instance}/records.
   Запрос на чтение потребителем `my_consumer` из группы `my_group` сообщений из топика, на который была подключена подписка, выглядит следующим образом:

```
$ curl -X GET -H "Accept: application/vnd.kafka.json.v2+json" \
      http://hostname:8082/consumers/my_group/instances/my_consumer/records
```

   где `H "Accept: application/vnd.kafka.json.v2+json"` – определяет пользовательский заголовок, указывающий тип контента, который клиент может принять.

   Результат запроса при успешном чтении сообщений:

```
[{"topic":"my_topic2","key":null,"value":{"name":"Helen"},"partition":0,"offset":0}]
```

   Прочитана пара ключ/значение `{"name":"Helen"}` в JSON-формате, которая была записана в топик.

4. Закрыть потребителя можно при помощи формы запроса DELETE /consumers/{string:group_name}/instances/{string:instance}.
   Запрос на закрытие потребителя `my_consumer` из группы `my_group` выглядит следующим образом:

```
$ curl -X DELETE -H "Content-Type: application/vnd.kafka.v2+json" \
      http://hostname:8082/consumers/my_group/instances/my_consumer
```

   Потребитель удален и больше не может читать сообщения.

Получение данных о топиках

Список топиков

Для получения списка топиков используется форма запроса GET /topics.

Запрос на получение списка топиков кластера ADS, в котором сервис Kafka REST Proxy установлен на хосте `hostname`, выглядит следующим образом:

```
$ curl -X GET http://hostname:8082/topics/
```

В результате запроса показаны все топики, существующие в кластере:

```
["my_topic","_schemas"]
```

Конфигурационные параметры топиков

Для получения данных о конфигурационных параметрах топика используется форма запроса GET /topics/{string:topic_name}.

Запрос на получение параметров топика `my_topic` выглядит следующим образом:

```
$ curl -X GET http://hostname:8082/topics/my_topic
```

В результате запроса отображаются все параметры запрашиваемого топика:

```
{"name":"my_topic","configs":{"compression.type":"producer","leader.replication.throttled.replicas":"","message.downconversion.
enable":"true","min.insync.replicas":"1","segment.jitter.ms":"0","cleanup.policy":"delete","flush.
ms":"9223372036854775807","follower.replication.throttled.replicas":"","segment.bytes":"1073741824","retention.
ms":"604800000","flush.messages":"9223372036854775807","message.format.version":"2.0-IV1","max.compaction.lag.
ms":"9223372036854775807","file.delete.delay.ms":"60000","max.message.bytes":"1048588","min.compaction.lag.ms":"0","message.
timestamp.type":"CreateTime","preallocate":"false","min.cleanable.dirty.ratio":"0.5","index.interval.bytes":"4096","unclean.leader.
election.enable":"false","retention.bytes":"-1","delete.retention.ms":"86400000","segment.ms":"604800000","message.timestamp.
difference.max.ms":"9223372036854775807","segment.index.bytes":"10485760"},"partitions":[{"partition":0,"leader":1001,"replicas":
[{"broker":1001,"leader":true,"in_sync":true}]}]}
```

Партиции

Для получения сведений о партициях топика используется форма GET /topics/{string:topic_name}/partitions.

Запрос на получение сведений о партициях топика `my_topic` выглядит следующим образом:

```
$ curl -X GET http://hostname:8082/topics/my_topic/partitions
```

В результате запроса отображаются сведения о партициях топика:

```
[{"partition":0,"leader":1001,"replicas":[{"broker":1001,"leader":true,"in_sync":true}]}]
```

Использование kafka-rest-security

Ольга Семина

Содержание

Назад к содержанию

Обзор

Предварительные требования

Включение модуля kafka-rest-security

Пример работы с аутентифицированным пользователем

Обзор

По умолчанию пользователь Kafka REST Proxy может аутентифицироваться в Kafka только с одним предопределенным принципалом, созданным при включении Kerberos.

Модуль `kafka-rest-security` предоставляет возможность проходить аутентификацию в Kafka с разными принципалами при подключении к Kafka через Kafka REST Proxy.

Запрос фильтруется через модуль `kafka-rest-security`, чтобы получить имя пользователя и сопоставить его со списком предопределенных пользователей для аутентификации в Kafka.

Если имя пользователя Kafka REST Proxy совпадает с одним из пользователей Kafka, новый контекст безопасности создается с применением этого принципала и используется для аутентификации в Kafka.

Предварительные требования

Для включения модуля `kafka-rest-security` должны быть выполнены следующие условия:

- В кластере ADS включена аутентификация про протоколу Kerberos с использованием любой из систем аутентификации:
  - MIT KDC
  - Active Directory
  - FreeIPA
  - Samba
- В системе аутентификации создан пользователь, имеющий принципал и keytab-файл для аутентификации в Kafka.
- В кластере ADS для сервиса Kafka REST Proxy настроена аутентификация, например, basic или при помощи файла *jaas*, создан пользователь, и ему назначена роль.

Включение модуля kafka-rest-security

Для включения модуля на странице конфигурирования сервиса Kafka REST Proxy выполните:

1. Переведите в активное состояние переключатель ***Show advanced***.
2. Откройте шаблон ***JAAS template***, где укажите в разделе ***KafkaClient*** данные пользователей:
   - ***userNames*** — опция записи имени пользователя, используется для сопоставления субъекта Kerberos субъекту *kafka-rest*. Имя пользователя должно совпадать с именем, введенным при создании нового пользователя во время basic-аутентификации. Могут быть указаны несколько имен пользователей через пробел.
   - ***keyTab*** — опция для указания keytab-файла субъекта Kerberos.
   - ***principal*** — опция для указания принципала субъекта Kerberos.

   Для каждого пользователя должен быть указан весь раздел полностью. Например, ниже приведены две записи для пользователя, в первой из них для пользователя Kafka REST Proxy `username` сопоставлены параметры субъекта Kerberos, а во второй не сопоставлены:

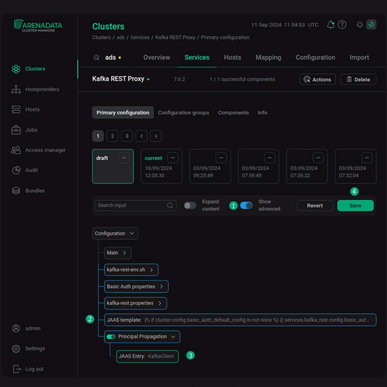
```
KafkaClient {
    com.sun.security.auth.module.Krb5LoginModule required
    useKeyTab=true
    storeKey=true
    useTicketCache=false
    serviceName="kafka"
    userNames="username"
    keyTab="/etc/security/keytabs/kafka-rest.service.keytab"
    principal="kafka-rest/<hostname>@<realm>";

    com.sun.security.auth.module.Krb5LoginModule required
    useKeyTab=true
    storeKey=true
    useTicketCache=false
    serviceName="kafka"
    keyTab="/etc/security/keytabs/kafka.service.keytab"
    principal="kafka/<hostname>@<realm>";
};
```

ПРИМЕЧАНИЕ

- Подробнее о шаблоне ***JAAS template*** см. в статье Настройка пользовательского jaas.conf.
- Ниже указаны настраиваемые в JAAS-файле готовые методы аутентификации, которые можно использовать с модулем `kafka-rest-security` для сопоставления участников (имен пользователей) Kafka REST Proxy с участниками Kafka:
  - JDBCLoginModule
  - PropertyFileLoginModule
  - DataSourceLoginModule
  - LdapLoginModule
- Совместно с вышеперечисленными методами может быть использована аутентификация mTLS. Данный метод аутентификации не может быть использован отдельно, так как он не предоставляет сведения о принципале REST.

3. Переведите в активное состояние переключатель ***Principal Propagation***. В поле ***JAAS Entry*** укажите раздел пользовательского файла ***JAAS template***, в котором содержатся данные новых пользователей.
4. Сохраните настройки, нажав ***Save***.



Настройка kafka-rest-security

После изменения параметров при помощи интерфейса ADCM перезагрузите сервис Kafka REST Proxy. Для этого примените действие ***Restart***, нажав на иконку в столбце ***Actions***.

Убедитесь, что в конфигурационном файле /etc/kafka-rest/kafka-rest.properties появились параметры:

```
kafka.rest.resource.extension.class=io.arenadata.kafkarest.security.KafkaRestSecurityResourceExtension
kafka.rest.auth.impersonation.enabled=true
kafka.rest.auth.impersonation.jaas.entry=KafkaClient
```

где:

- `kafka.rest.resource.extension.class` — список классов, которые будут использоваться для внедрения пользовательских ресурсов;
- `kafka.rest.auth.impersonation.enabled` — включает распознавание имен пользователей в Kafka REST Proxy;
- `kafka.rest.auth.impersonation.jaas.entry` — указывает раздел JAAS-файла для распознавания имен.

Метод аутентификации mTLS включается при помощи добавления пользовательского свойства в файл *kafka-rest.properties* (с использованием поля ***Add key, value*** на странице конфигурирования сервиса Kafka REST Proxy):

- ***Enter field name*** — `ssl.client.authentication`
- ***Enter field value*** — `REQUIRED`

ВНИМАНИЕ

Для Kafka REST необходимо настроить SSL до включения аутентификации по методу mTLS.

mTLS необходимо объединить с одним из модулей входа в систему JAAS для использования с `kafka-rest-security`.

Список готовых модулей, которые можно настроить в шаблоне JAAS:

- org.eclipse.jetty.security.jaas.spi.JDBCLoginModule
- org.eclipse.jetty.security.jaas.spi.PropertyFileLoginModule
- org.eclipse.jetty.security.jaas.spi.DataSourceLoginModule
- org.eclipse.jetty.security.jaas.ldap.LdapLoginModule

Пример работы с аутентифицированным пользователем

Ниже приведен запрос для записи пары ключ/значение `{"foo":"bar"}` в топик `test` с аутентификацией пользователя, где `<username:password>` — имя пользователя и пароль, введенные при создании нового пользователя во время basic-аутентификации:

```
$ curl -u <username:password> -X POST -H "Content-Type: application/vnd.kafka.json.v2+json" --data '{"records":[{"value":{"foo":"bar"}}]}' "http://hostname:8082/topics/test"
```

ПРИМЕЧАНИЕ

Подробнее о работе с Kafka REST Proxy при помощи cURL описано в статье Пример использования Kafka REST Proxy.

Пример использования ksqlDB

Alex Services

Содержание

Назад к содержанию

Запись событий в топик Kafka

Запуск ksqlDB

Создание потока из топика Kafka

Управление данными потока

- Просмотр данных
- Подсчет событий
- Выборка по названию

Запись из ksqlDB в топик Kafka

В статье описан пример управления данными о событиях на сайте (например, Wikipedia) при помощи сервиса ksqlDB.

Запись событий в топик Kafka

Для записи 10 последних событий с сайта можно использовать Python-скрипт, выполняющий:

1. Определение параметров конфигурации для производителя (producer) Kafka.
2. Определение параметров конфигурации для работы с Wikipedia API.
3. Создание производителя (producer) Kafka.
4. Получение последних изменений из API Wikipedia (данные о последних 10 отредактированных статьях).
5. Запись событий в топик Kafka в JSON-формате.
6. Закрытие производителя (producer) Kafka.

ПРИМЕЧАНИЕ

Для работы с данным Python-скриптом необходимо предварительно установить пакеты:

- requests
- kafka-python

Для записи событий в топик Kafka необходимо:

1. Создать файл скрипта:

```
$ sudo vim /tmp/events.py
```

2. Заполнить events.py текстом скрипта, приведенным ниже, заменив индивидуальные значения:

- `bootstrap_servers` – имя хоста для подключения к серверу Kafka.
- `topic_name` – имя топика для записи событий.

events.py

```
import requests
from kafka import KafkaProducer
import json

# Kafka producer configuration
bootstrap_servers = 'ser-test-4.ru-central1.internal:9092'
topic_name = 'wikipedia-topic'

# Wikipedia API configuration
api_url = 'https://en.wikipedia.org/w/api.php'
params = {
    'action': 'query',
    'format': 'json',
    'list': 'recentchanges',
    'rcprop': 'user|title|timestamp',
    'rclimit': '10'
}

# Create a Kafka producer
producer = KafkaProducer(bootstrap_servers=bootstrap_servers,
                         value_serializer=lambda v: json.dumps(v).encode('utf-8'))

# Fetch recent changes from Wikipedia API and produce messages to Kafka
response = requests.get(api_url, params=params)
data = response.json()
recent_changes = data['query']['recentchanges']

for change in recent_changes:
    event = {
        'user_id': change['user'],
        'page_title': change['title'],
        'event_type': 'edit',
        'timestamp': change['timestamp']
    }
    producer.send(topic_name, value=event)
    print("Produced event:", event)

# Close the Kafka producer
producer.close()
```

3. Дать разрешение на выполнение скрипта:

```
$ sudo chmod +x /tmp/events.py
```

4. Запустить скрипт:

```
$ python /tmp/events.py
```

В результате правильного выполнения скрипта выводится события в JSON-формате.

Пример записанных событий с сайта

```
{'Produced event': {'page_title': 'user:199.203.61.245', 'user_id': 'U1A7Provy8pK', 'event_type': 'edit', 'timestamp':
'2023-05-31T15:17:48Z'}}
{'Produced event': {'page_title': 'user:178.221.183.209', 'user_id': 'U1A7Provy8pK', 'event_type': 'edit', 'timestamp':
'2023-05-31T15:17:48Z'}}
{'Produced event': {'page_title': 'user:Carlos Alcaraz career statistics', 'user_id': 'U1A7DoUeeCl1on', 'event_type': 'edit',
'timestamp': '2023-05-31T15:17:47Z'}}
{'Produced event': {'page_title': 'user:178.4.164.242', 'user_id': 'U1A7Provy8pK', 'event_type': 'edit', 'timestamp':
'2023-05-31T15:17:47Z'}}
{'Produced event': {'page_title': 'user:45.238.251.170', 'user_id': 'U1A7Provy8pK', 'event_type': 'edit', 'timestamp':
'2023-05-31T15:17:47Z'}}
{'Produced event': {'page_title': 'user:Id.8.67 79', 'user_id': 'U1A7Provy8pK', 'event_type': 'edit', 'timestamp': '2023-
05-31T15:17:47Z'}}
{'Produced event': {'page_title': 'user:Para fanoiga on screen and stage', 'user_id': 'U1Cheiznaq3x', 'event_type': 'edit',
'timestamp': '2023-05-31T15:17:46Z'}}
{'Produced event': {'page_title': 'user:Personal health record', 'user_id': 'U1John Of Reading', 'event_type': 'edit',
'timestamp': '2023-05-31T15:17:46Z'}}
{'Produced event': {'page_title': 'user:199.204.180.128', 'user_id': 'U1A7Provy8pK', 'event_type': 'edit', 'timestamp':
'2023-05-31T15:17:46Z'}}
{'Produced event': {'page_title': 'user:Tech7solon', 'user_id': 'U1Tech7solon', 'event_type': 'edit', 'timestamp': '2023-05-
31T15:17:44Z'}}
```

Для просмотра данных, записанных в топик Kafka, можно прочитать данные из топика Kafka при помощи командной строки.

Запуск ksqlDB

После установки ksqlDB запуск сервиса осуществляется при помощи команды:

```
$ sudo ksql http://<hostname>:8088
```

где `<hostname>` – имя хоста, на котором установлен сервис ksqlDB.

Сервер запущен:

```
                  ===========================================
                  =       _              _ ____  ____       =
                  =      | | _____  __ _| |  _ \| __ )      =
                  =      | |/ / __|/ _` | | | | |  _ \      =
                  =      |   <\__ \ (_| | | |_| | |_) |     =
                  =      |_|\_\___/\__, |_|____/|____/      =
                  =                   |_|                   =
                  =  Event Streaming Database purpose-built =
                  =        for stream processing apps       =
                  ===========================================

Copyright 2017-2021 Confluent Inc.

CLI v6.2.1, Server v6.2.1 located at http://ser-test-4.ru-central1.internal:8088
Server Status: RUNNING

Having trouble? Type 'help' (case-insensitive) for a rundown of how things work!

ksql>
```

Запуск функции ksqlDB производится в строке `ksql>`.

Создание потока из топика Kafka

Для комментирования данных, записанных в топик Kafka, при помощи ksqlDB необходимо создать поток.

Поток (stream) – секционированная, неизменяемая, доступная только для добавления коллекция, которая представляет ряд фактов о событиях или записях.

Запрос для создания потока `wikipedia_events` для топика Kafka `wikipedia-topic` выглядит следующим образом:

```
CREATE STREAM wikipedia_events (\
    user_id VARCHAR,\
    page_title VARCHAR,\
    event_type VARCHAR,\
    timestamp VARCHAR\
) WITH (\
    kafka_topic='wikipedia-topic',\
    value_format='json',\
    key_format='sone'\
);
```

где задано:

- наименование потока;
- имена столбцов потока.

Поток создан:

```
Message
----------------
Stream created
----------------
```

Управление данными потока

При помощи различных запросов можно извлекать данные из созданного потока.

ПРИМЕЧАНИЕ

Для того, чтобы поток был доступен с начала, а не с самого последнего смещения, можно установить в работающей строке параметр, определяющий, с какого смещения требуется читать поток:

```
SET 'auto.offset.reset' = 'earliest';
```

Без этой установки для отработки приведенных ниже примеров необходимо после запуска функции ksqlDB перезапустить данные в потоке (например, заново выполнить скрипт, описанный выше).

Просмотр данных

Просмотреть все содержимое потока `wikipedia_events` можно при помощи запроса с применением оператора `SELECT` без указания условий выборки:

```
SELECT * FROM wikipedia_events EMIT CHANGES;
```

где `EMIT CHANGES` – указывает на то, что следует выводить все происходящие изменения.

Результат запроса приведен ниже:

```
+------------------------+------------------------+------------------------+------------------------+
|USER_ID                 |PAGE_TITLE              |EVENT_TYPE              |TIMESTAMP               |
+------------------------+------------------------+------------------------+------------------------+
|M. Armando              |Ibiza Guerra            |edit                    |2023-05-29T10:56:21Z    |
|Depressedpar            |Talk:Bring It Back (Mates of Stat|edit           |2023-05-29T10:56:28Z    |
|                        |te album)               |edit                    |2023-05-29T10:56:28Z    |
|Yeji 220.255.5          |Visitor Kuhio           |edit                    |2023-05-29T10:56:30Z    |
|Jensonsantos            |Category:Research       |edit                    |2023-05-29T10:56:32Z    |
|Strumatankuncluvre      |IRN XB                  |edit                    |2023-05-29T10:56:32Z    |
|Yeji 220.255.156        |Category:1966 establishments in |edit            |2023-05-29T10:56:15Z    |
|                        |South Africa            |edit                    |2023-05-29T10:56:15Z    |
|Yeji 220.255.156        |BBC Air Navigation School SAAF  |edit                    |2023-05-29T10:56:15Z    |
|Ibaiz or Nehman Nisar   |User:Ibaiz or Nehman Nisar      |edit                    |2023-05-29T10:56:15Z    |
|Dilka                   |2023 Cannes Film Festival       |edit                    |2023-05-29T10:56:14Z    |
|Super Dromeosaurus      |Re:[verse]. Tolerman            |edit                    |2023-05-29T10:56:14Z    |
```

Если еще раз запустить скрипт, описанный выше, для того же потока, можно увидеть, что в поток поступают новые данные.

Подсчет событий

Для выхода количества редакций статей из выборки, записанной в поток, можно использовать операторы `SELECT` и `COUNT` с сохранением результатов в новом потоке `EDIT_COUNT`.

Для ее подсчета можно дополнительно исключить записей в поток Kafka, изменив в скрипте количество возвращаемых изменений (`"rclimit": 10` – до 100 и более), а также запустить скрипт.

Запрос для подсчета количества редакций последних 10 статей в потоке `wikipedia-events` и сохранения результатов в новом потоке `EDIT_COUNT` выглядит следующим образом:

```
SELECT page_title, COUNT(*) AS edit_count
FROM wikipedia_events
GROUP BY page_title
EMIT CHANGES
LIMIT 10;
```

где применена функция GROUP BY.

Результат запроса приведен ниже:

```
+------------------------+------------------------+
|PAGE_TITLE              |EDIT_COUNT              |
+------------------------+------------------------+
|Category:Redirects from long names |1            |
|Classe-fiction: The Darnassus Years |1           |
|a Complete Coverage of the Genre Regal |1        |
|Cities  - From 1914a Throughout 1916a |1        |
|Ben Anderson (journalist) |1                    |
|User talk:179.142.81.line |1                    |
|Wikipedia:Articles for creation/Redire|1         |
|cts and categories |1                           |
|2023 North Indian Ocean cyclone season|1        |
|Category:Wikipedia articles needing c|1          |
|Category:AfC submissions by date/04 Jun|1        |
|ne 2023                 |                       |
|Category:Redirect-Class AFC articles |1         |
|Category:Unprintworthy redirects |1              |
|List Hanshed            |                       |
+------------------------+------------------------+
```

Выборка по названию

Для осуществления выборки статей, в названии которых содержится `Category:`, используется оператор `SELECT` и функция `WHERE`.

```
SELECT page_title,
timestamp
FROM wikipedia_events
WHERE page_title LIKE 'Category:%'
EMIT CHANGES;
```

Результат запроса приведен ниже:

```
+------------------------+------------------------+
|PAGE_TITLE              |TIMESTAMP               |
+------------------------+------------------------+
|Category:Redirects from long names |2023-05-01T07:21:57Z|
|Category:Accepted AFC submissions |2023-05-01T07:21:57Z|
|Category:AfC submissions by date/04 June 2023 |2023-05-01T07:21:57Z|
|Category:Redirect-Class AFC articles |2023-05-01T07:21:58Z|
|Category:Unprintworthy redirects |2023-05-01T07:21:58Z|
```

Запись из ksqlDB в топик Kafka

Для записи последней выборки в новый топик Kafka `new_topic` в формате JSON можно создать новый поток `wikipedia_new`:

```
CREATE STREAM wikipedia_new
  WITH ( kafka_topic='new_topic', value_format='json')
  AS SELECT
  page_title,
  timestamp
  FROM wikipedia_events
  WHERE page_title LIKE 'Category:%'
  EMIT CHANGES;
```

Поток создан:

```
 Message
----------------------------
 Created query with ID CSAS_WIKIPEDIA_NEW_191
----------------------------
```

В строке `ksql>` также возможно запустить команду для чтения содержимого, записанных в топик `new_topic`:

```
print new_topic;
```

Результат запроса приведен ниже:

```
Key format: \_(-/-)_/ - no data processed
Value format: JSON or KAFKA_STRING
rowtime: 2023/06/01 07:21:58 522 Z, key: <null>, value: {"PAGE_TITLE":"Category:Redirects from long names","TIMEST
AMP":"2023-06-01T07:21:57Z"}, partition: 0
rowtime: 2023/06/01 07:21:58 522 Z, key: <null>, value: {"PAGE_TITLE":"Category:Accepted AFC submissions","T
IMESTAMP":"2023-06-01T07:21:57Z"}, partition: 0
rowtime: 2023/06/01 07:21:58 522 Z, key: <null>, value: {"PAGE_TITLE":"Category:AfC submissions by date/04 June
2023","TIMESTAMP":"2023-06-01T07:21:57Z"}, partition: 0
rowtime: 2023/06/01 07:21:58 522 Z, key: <null>, value: {"PAGE_TITLE":"Category:Redirect-Class AFC articles","TIMESTAMP":"2023-
06-01T07:21:58Z"}, partition: 0
rowtime: 2023/06/01 07:21:58 522 Z, key: <null>, value: {"PAGE_TITLE":"Category:Unprintworthy redirects","TIMESTAMP":"2023-06-
01T07:21:58Z"}, partition: 0
```

В топик записаны сообщения в соответствии с выполненной выборкой.

Обзор UI NiFi Server

Содержание

Подключение к NiFi Server
Главное окно
- Панель инструментов
- Строка состояния
- Палитра операций
- Палитра навигации

Поиск
Глобальное меню

Назад к содержанию

Подключение к NiFi Server

После установки сервера NiFi по адресам, которые будут выведены в журнал, есть по умолчанию и основное подключение NiFi Server. Если же нужно перейти сервер, отличный от указанного в журнале, то можно воспользоваться функционалом Controller Settings, который находится в глобальном меню, или сразу ввести URL нужного сервера NiFi.

Пример интерфейса NiFi Server

После успешной авторизации NiFi откроет главное окно, для старта нам потребуется панель инструментов, а также корневая группа, в которой мы можем создать процессоры или другие элементы NiFi.

Главное окно

Это наш основной экран! Мы можем видеть всю вкладку и последовательность интерфейса NiFi. По умолчанию вкладка пользовательского интерфейса NiFi также имеет два дополнительных:

Панель инструментов и интерфейса NiFi

Изначально пользовательский интерфейс NiFi также содержит три частей, а именно панель строки состояния и панель.

Панель инструментов

Панель инструментов содержит следующие элементы:

- Компоненты (Processor) | Компонент, который является основным компонентом потока. Существуют несколько типов процессоров.
- Группа ввода (Input Port) | Этот компонент предоставляет способ получения данных из группы процессов.
- Группа вывода (Output Port) | Этот компонент предоставляет способ передачи данных из группы процессов.
- Группа (Process Group) | Группа, в которой можно размещать другие компоненты. Узлы, упрощать понимание и избегать беспорядка.
- Удаленная группа ввода (Remote Process Group) | Этот компонент представляет собой группу ввода на удаленном экземпляре NiFi.
- Удаленная группа вывода (Remote Process Group) | Этот компонент представляет собой группу вывода на удаленном экземпляре NiFi.
- Шаблоны (Template) | Шаблоны позволяют пользователям повторно использовать и обмениваться потоками данных.
- Метка (Label) | Позволяет пользователям создавать комментарии для документирования потока.

Строка состояния

Строка состояния находится под панелью инструментов.

Строка состояния

Строка состояния содержит следующие элементы:

- Количество потоков, которые в настоящее время работают.
- Количество потоков, которые остановлены.
- Количество потоков, которые недействительны.
- Количество потоков, которые имеют предупреждения.
- Количество потоков, которые находятся в состоянии ошибки.
- Количество потоков, которые находятся в состоянии обновления.

Строка состояния также содержит следующую информацию:

- Общее количество потоков, которые были развернуты.
- Количество подключенных удаленных групп процессов.
- Количество отправленных пакетов FlowFile.
- Количество полученных пакетов FlowFile.
- Количество пакетов FlowFile, которые были переданы.
- Количество пакетов FlowFile, которые были получены.

Строка состояния также содержит следующие кнопки:

- Кнопка обновления, которая обновляет пользовательский интерфейс.

Кнопка обновления

- Кнопка настройки, которая открывает окно настройки.

Кнопка настройки

- Кнопка справки, которая открывает окно справки.

Кнопка справки

- Кнопка выхода, которая выходит из текущего сеанса.

Кнопка выхода

- Кнопка входа, которая открывает окно входа.

Кнопка входа

- Кнопка информации о пользователе, которая открывает окно информации о пользователе.

Кнопка информации о пользователе

Обзор UI NiFi Registry

Ольга Семме

Содержание

Назад к содержанию

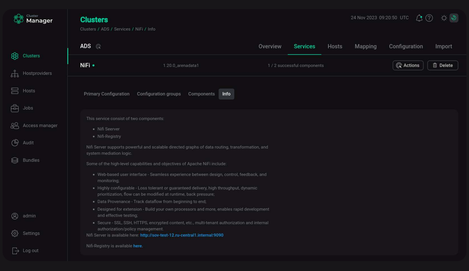
Подключение к NiFi Registry

Главное окно

Подключение к NiFi Registry

После установки сервиса NiFi в адресной строке браузера необходимо ввести адрес хоста, на котором установлен компонент NiFi Registry. В качестве номера порта следует указать значение параметра ***Nifi Registry UI port*** из раздела ***Main***, определенное на этапе конфигурирования сервиса NiFi при установке кластера ADS.

Другой способ подключения к интерфейсу NiFi Registry — перейти по ссылке, указанной на вкладке ***Info*** меню сервиса NiFi.

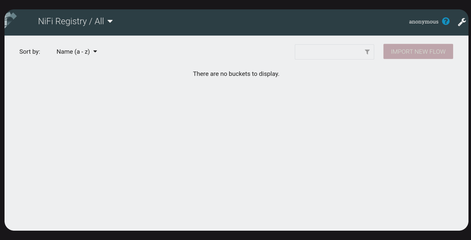


Переход к интерфейсу NiFi Registry

Пользовательский интерфейс NiFi Registry предоставляет инструменты для хранения и управления общими ресурсами в одном или нескольких экземплярах NiFi и/или MiNiFi.

Главное окно

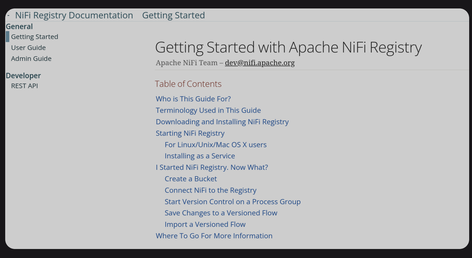
После входа в пользовательский интерфейс NiFi Registry пользователь видит главное окно. По умолчанию вход в пользовательский интерфейс NiFi Registry доступен без аутентификации.



Главное окно пользовательского интерфейса NiFi Registry

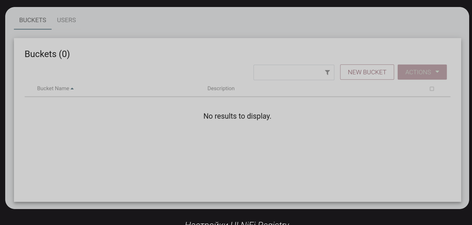
В главном окне NiFi Registry отображаются следующие элементы:

- anonymous — имя текущего пользователя. Отображается в правом верхнем углу. Если аутентификация не настроена, отображается имя `anonymous`.
- — иконка справки, открывает доступ к документации NiFi Registry. Отображается в правом верхнем углу.



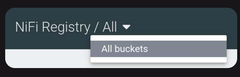
Документация NiFi Registry

- — иконка настройки, открывает окно настроек UI NiFi Registry. Отображается в правом верхнем углу.



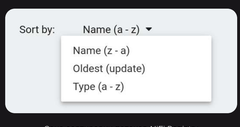
Настройки UI NiFi Registry

- Меню "Buckets" – позволяет пользователю отображать потоки (Flows) в зависимости от того, в какой корзине (Bucket) они содержатся. Находится в верхней левой части экрана.



Меню Buckets

- Зона сортировки потоков — позволяет сортировать потоки по имени, типу и времени обновления. Находится в левой части экрана под меню "Buckets".

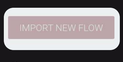


Зона сортировки потоков NiFi Registry

- Строка фильтра — позволяет установить фильтр для отображения групп потоков, объединеных общим обозначением. Находится в правой части экрана.

Строка фильтра

- Кнопка ***Import new flow*** — позволяет импортировать поток NiFi Registry. Находится в правой части экрана.



Кнопка импорта потоков

ПРИМЕЧАНИЕ

Для получения более подробной информации о пользовательском интерфейсе Apache NiFi Registry обратитесь к статье NiFi Registry User Interface.

Аутентификация в сервисе NiFi

Содержание

Назад к содержанию

- Предварительные требования для включения аутентификации
- Установка параметров конфигурации
  - Необходимые параметры для настройки SSL
  - Необходимые параметры для настройки LDAP
- Проверка включения аутентификации NiFi Server
- Проверка включения аутентификации NiFi Registry
- Выключение аутентификации
- Изменение данных в NiFi Initial Admin

Текстовые материалы подготовлены в рамках [Курса по Apache NiFi](). Если у вопросов, [пишите в Telegram](). Видеоверсия [доступна на YouTube]().

Apache NiFi по умолчанию работает в режиме [HTTP](). Для включения [HTTPS]() необходимо выполнить настройку сервиса на использование [SSL]() и настроить [аутентификацию]() пользователей.

Включение аутентификации выполняется в несколько этапов:

1. Установка параметров конфигурации
2. Проверка включения аутентификации NiFi Server
3. Проверка включения аутентификации NiFi Registry

Для включения аутентификации в NiFi можно использовать несколько [провайдеров](). Для сервиса необходимо указать только **один** активный провайдер для доступа к веб-интерфейсу.

Предварительные требования для включения аутентификации

Внимание!

Настройка SSL для NiFi была выполнена ранее в [этой статье]().

Задача: настроить аутентификацию пользователей в NiFi с использованием [LDAP-провайдера]() и [Single User Authorizer]().

Для этого:

1. Настроим [LDAP-сервер]() для NiFi, где будут храниться учетные записи пользователей.
2. Настроим [SSL]() на стороне NiFi, чтобы данные для аутентификации передавались по защищенному каналу.
3. Включим аутентификацию пользователей в NiFi и настроим [авторизацию]() доступа к ресурсам.

Для выполнения задания нам потребуется [Docker]() и [Docker Compose]().

Для работы примера нам потребуется:

- **Active Directory** — это Windows-сервис и набор служб для централизованного управления ресурсами сети.
- **LDAP-сервер** — это протокол, который позволяет приложениям обращаться к службам Active Directory, которые может содержать пользователей, группы и компьютеры.
- **Вспомогательный контейнер для работы клиентов** (**phpldapadmin**) — это веб-интерфейс для управления LDAP-сервером.

Перед тем, как использовать Active Directory, нужно настроить его. Для этого:

- Создать [Forest]() — это самый верхний уровень иерархии в службе Active Directory, который может содержать пользователей, группы и компьютеры.
- Создать [Domain]() — это группа объектов, которые имеют общую базу данных.
- Создать [Organizational Unit]() — это объекты, которые должны быть назначены пользователю или группе.
- Создать [Group]() — это группа пользователей.
- Создать [User]() — это учетная запись пользователя.

Перед тем, как использовать LDAP-сервер, нужно настроить его. Для этого:

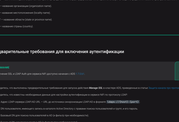
- Задать [DOMAIN]() — это группа объектов, которые имеют общую базу данных.
- Задать [ORGANIZATION]() — это объекты, которые должны быть назначены пользователю или группе.
- Задать [ADMIN_PASSWORD]() — это группа пользователей.

Внимание!

Для работы примера нам потребуется [Docker]() и [Docker Compose]().

Установка параметров конфигурации

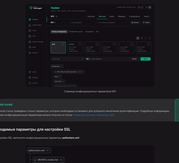
Также убедимся, что [файл конфигурации]() `nifi.properties` настроен корректно:



Настройка конфигурационного файла nifi.properties

Необходимые параметры для настройки SSL

Для настройки SSL достаточно конфигурационных параметров:

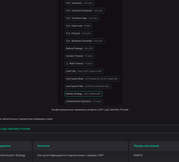


Настройка конфигурационного файла authorizers.xml

Параметр	Описание	Пример заполнения
`nifi.web.https.host`	Имя хоста, на котором будет доступен NiFi	`localhost`
`nifi.web.https.port`	Порт, на котором будет доступен NiFi	`8443`
`nifi.security.keystore`	Путь к файлу keystore	`./conf/keystore.jks`
`nifi.security.keystoreType`	Тип keystore	`JKS`
`nifi.security.keystorePasswd`	Пароль keystore	`changeme`
`nifi.security.keyPasswd`	Пароль ключа	`changeme`
`nifi.security.truststore`	Путь к файлу truststore	`./conf/truststore.jks`
`nifi.security.truststoreType`	Тип truststore	`JKS`
`nifi.security.truststorePasswd`	Пароль truststore	`changeme`
`nifi.security.user.authorizer`	Авторизатор, который будет использоваться для авторизации пользователей	`single-user-authorizer`
`nifi.security.user.login.identity.provider`	Провайдер, который будет использоваться для аутентификации пользователей	`ldap-provider`

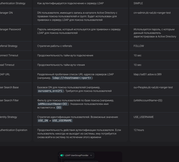
Необходимые параметры для настройки LDAP

Для настройки параметров LDAP достаточно настроить **LDAP Login Identity Provider** и **LDAP UserGroupProvider**:



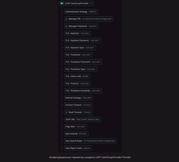
Настройка конфигурации провайдера LDAP Login Identity Provider

Описание обязательных параметров приведено ниже:



LDAP Login Identity Provider

Параметр	Описание	Пример заполнения
Authentication Strategy	Как аутентифицируется подключение к серверу LDAP	`START_TLS`
Manager DN	Полный DN пользователя, с которым будет выполнен привязка к серверу LDAP для поиска пользователей	`cn=admin,dc=nifi-ldap,dc=local`
Manager Password	Пароль пользователя, с которым будет выполнен привязка к серверу LDAP для поиска пользователей	`Alt1ssaAyYJ_bpyOL`
Referral Strategy	Стратегия работы с referrals	`FOLLOW`
Connect Timeout	Предельное время ожидания при подключении	`10 sec`
Read Timeout	Предельное время ожидания при чтении	`10 sec`
LDAP URL	Список URL-адресов LDAP-серверов, разделенных пробелами. Должны использоваться протоколы `ldap://` или `ldaps://`	`ldap://ldap1 ldaps://ldap2`
User Search Base	Базовый DN для поиска пользователей. Не должен содержать `OU=Domain Controllers`	`ou=People,dc=nifi-ldap,dc=local`
User Search Filter	Фильтр для поиска пользователей. Должен содержать `{0}`	`(&(objectClass=inetOrgPerson)(uid={0}))`
Identity Strategy	Как будет извлекаться имя пользователя. Возможные значения: `USE_DN` и `USE_USERNAME`	`USE_USERNAME`
Authentication Expiration	Время, в течение которого аутентификация пользователя считается действительной	`12 hours`



LDAP UserGroupProvider

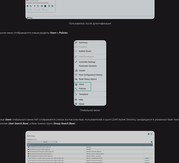
Описание обязательных параметров приведено ниже:

Параметр	Описание	Пример заполнения
Authentication Strategy	Как аутентифицируется подключение к серверу LDAP	`START_TLS`
Manager DN	Полный DN пользователя, с которым будет выполнен привязка к серверу LDAP для поиска пользователей. Для доступа к группам `nifi-admins` и `nifi-users` должен использоваться DN администратора	`cn=admin,dc=nifi-ldap,dc=local`
Manager Password	Пароль пользователя, с которым будет выполнен привязка к серверу LDAP для поиска пользователей	`Alt1ssaAyYJ_bpyOL`
Referral Strategy	Стратегия работы с referrals	`FOLLOW`
Connect Timeout	Предельное время ожидания при подключении	`10 sec`
Read Timeout	Предельное время ожидания при чтении	`10 sec`
LDAP URL	Список URL-адресов LDAP-серверов, разделенных пробелами. Должны использоваться протоколы `ldap://` или `ldaps://`	`ldap://ldap1 ldaps://ldap2`
Sync Interval	Предельное время ожидания между синхронизациями пользователей и групп. Рекомендуется `SYNC_INTERVAL`	`30 mins`
User Search Base	Базовый DN для поиска пользователей. Не должен содержать `OU=Domain Controllers`	`ou=People,dc=nifi-ldap,dc=local`
User Object Class	Класс объектов, представляющих пользователей	`person`
User Search Scope	Область для поиска пользователей	`ONE_LEVEL`
User Identity Attribute	Атрибут, содержащий имя пользователя. Рекомендуется `cn`	`cn`
Group Search Base	Базовый DN для поиска групп. Рекомендуется `OU=Groups`	`ou=Groups,dc=nifi-ldap,dc=local`
Group Object Class	Класс объектов, представляющих группы	`group`
Group Search Scope	Область для поиска групп пользователей	`ONE_LEVEL`
Group Member Attribute	Атрибут, содержащий DN участников группы	`member`

После основной настройки нажмите **Save**.

Проверка включения аутентификации NiFi Server

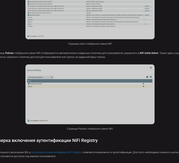
После успешного включения [HTTPS]() в [конфигурационном файле NiFi]() откроем браузер и зайдем по новому адресу:



Окно авторизации

Для первоначальной авторизации выполните вход под администратором NiFi, введите логин и пароль, указанные в документе для **NiFi Initial Admin**.

После успешной авторизации вы попадете в интерфейс NiFi:

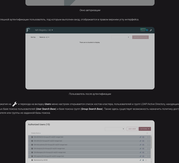


Интерфейс NiFi

Для проверки работы аутентификации выполните вход под другим пользователем, например, под **Admin**:

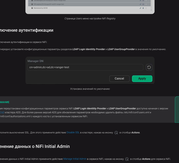
Интерфейс Admin

В нижнем углу экрана отображается надпись **Admin** и **Policies**:



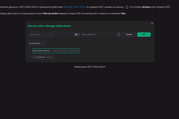
Интерфейс Admin

Чтобы открыть **Users** и **Groups**, нажмите на иконку. Вот отображаемый список групп:



Список групп

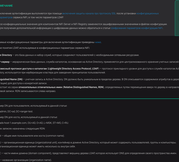
Чтобы открыть **Policies**, нажмите на иконку. Вот отображаемый список политик:



Список политик

Проверка включения аутентификации NiFi Registry

После успешного включения [HTTPS]() в [конфигурационном файле NiFi Registry]() откроем браузер и зайдем по новому адресу:



Окно авторизации

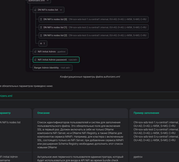
После успешной авторизации откроется окно NiFi Registry, отображается пустое окно:

Пустое окно

Чтобы открыть **Users** и **Groups**, нажмите на иконку. Вот отображаемый список групп:

Список групп

Чтобы открыть **Policies**, нажмите на иконку. Вот отображаемый список политик:

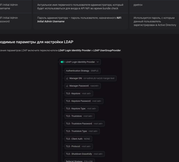


Список политик

Выключение аутентификации

Для выключения аутентификации нужно:

1. Поочередно установить в конфигурации параметры сервисов **LDAP Login Identity Provider** и **LDAP UserGroupProvider** в значение `Empty`.



Установка значения Empty для LDAP

2. Установить значение `single-user-authorizer` для параметра `nifi.security.user.authorizer`.

Установка значения single-user-authorizer

3. Выполнить перезапуск NiFi. Для этого выполните команду `./bin/nifi.sh restart` или нажмите на кнопку **Actions**.

Внимание!

Выключение аутентификации для [Registry]() выполняется аналогично.

Изменение данных в NiFi Initial Admin

Для изменения данных для [NiFi Initial Admin]() откройте [файл конфигурации]() `nifi.properties`, найдите строку `nifi.security.identity.mapping.pattern.dn` и скопируйте **Admin** для строки NiFi.

После выполнения данных операций нажмите **Run as admin** вверху окна. Вот отображаемое окно с новыми данными:

Изменение NiFi Initial Admin

Авторизация в пользовательском интерфейсе NiFi

Ольга Семина

Содержание

Назад к содержанию

NiFi Server
- Настройка политик доступа через глобальное меню NiFi
- Настройка политик доступа на уровне процессора

NiFi Registry

Авторизация в сервисе NiFi при помощи инструментов интерфейса осуществляется после аутентификации пользователя, имеющего учетную запись в Active Directory.

Назначить политики доступа возможно для пользователей Active Directory, находящихся в области поиска (User Search Base), заданной на этапе конфигурации параметров сервиса NiFi.

ПРИМЕЧАНИЕ

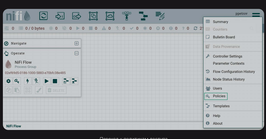
Все конфиденциальные значения для компонентов NiFi Server и NiFi Registry заменяются зашифрованными значениями в файлах конфигурации. Для получения дополнительной информации о шифровании данных можно обратиться к статье Шифрование параметров конфигурации NiFi.

NiFi Server

Настройка политик доступа через глобальное меню NiFi

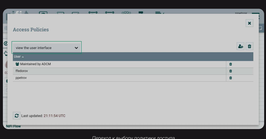
Для назначения политики доступа через глобальное меню интерфейса NiFi Server необходимо:

1. В глобальном меню перейти на страницу **Policies**.



Переход к политикам доступа

2. Открыть список политик доступа.



Переход к выбору политики доступа

3. Выбрать политику доступа из списка и нажать иконку назначения политики доступа.

Доступные политики доступа

ПРИМЕЧАНИЕ

Для получения подробной информации о политиках доступа в интерфейсе NiFi Server обратитесь к статье Access Policies NiFi.

4. Найти пользователя или группу для назначения политики, пользуясь строкой ***User Indentity***.

Поиск пользователя

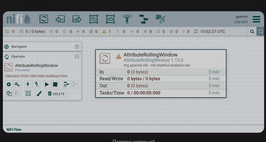
5. Выбрать пользователя и нажать **ADD**.
В результате в списке выбранной политики доступа появится новый пользователь или группа.

Список пользователей для политики доступа

Настройка политик доступа на уровне процессора

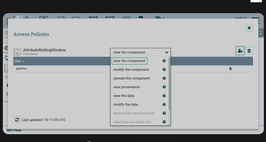
После настройки аутентификации из палитры операций становится доступна иконка , при помощи которой назначаются политики доступа на уровне выбранного процессора или группы. Для этого необходимо выполнить:

1. Выбрать необходимый процессор или группу и выделить его. Тогда на палитре операций появится название выбранного процессора.



Палитра операций

2. Нажать на иконку и открывшемся окне выбрать политику доступа из списка и нажать иконку назначения политики доступа.

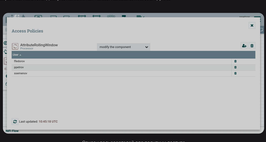


Политики доступа на уровне процессора

3. Найти пользователя или группу для назначения политики, пользуясь строкой ***User Indentity***.

Поиск пользователя

4. Выбрать пользователя и нажать **ADD**.
В результате в списке выбранной политики доступа появится новый пользователь или группа.



Список пользователей для политики доступа

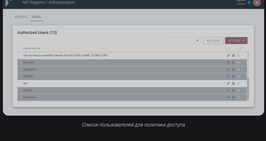
Удаление пользователя из политики доступа осуществляется при помощи иконки .

Для проверки созданной политики необходимо авторизоваться под пользователем, для которого назначалась политика.

NiFi Registry

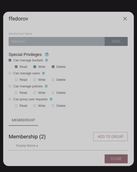
Для назначения политики доступа в интерфейсе NiFi Registry необходимо:

1. Кликнуть на иконку настроек , и перейти во вкладку **USERS**.



Список пользователей для политики доступа

2. Выбрать пользователя или группу и кликнуть на кнопку редактирования политики доступа .
3. Выбрать необходимые разрешения и нажать **CLOSE**.



Выбор необходимых разрешений

ПРИМЕЧАНИЕ

Для получения подробной информации о политиках доступа в интерфейсе NiFi Registry обратитесь к статье Access Policies NiFi Registry.

Для проверки созданной политики необходимо авторизоваться под пользователем, для которого назначалась политика.

Плагин Ranger NiFi

Ольга Семме

Содержание

Назад к содержанию

Предварительные требования

Организация взаимодействия кластеров по протоколу SSL

Активация плагина Ranger NiFi

Проверка активированного плагина Ranger NiFi

Предварительные требования

Предварительные требования для авторизации в NiFi при помощи Ranger:

1. Установлен кластер ADS с настроенной аутентификацией пользователя, имеющего учетную запись в Active Directory.
2. Установлен кластер ADPS.
3. Включен и настроен SSL (при помощи сертификатов SSL или сертификатов Active Directory) в кластере ADPS.
4. Для отключения конвертирования регистров имен пользователей и групп в NiFi следующим параметрам группы ***LDAP sync source for User synchronizer*** для сервиса Ranger кластера ADPS присвоено значение `none`:
   - ***ranger.usersync.ldap.groupname.caseconversion***
   - ***ranger.usersync.ldap.username.caseconversion***

   Для получения дополнительной информации про настройку синхронизации пользователей в Ranger обратитесь к статье Настройка Ranger User Sync.
5. Включена интеграция с кластером ADS в кластере ADPS.

ПРИМЕЧАНИЕ

- В данной статье описана авторизация при помощи ADPS с самоподписанными сертификатами SSL.
- Минимальные требования для включения авторизации в Ranger для сервиса NiFi:
  - ADPS версии 1.0.4.b3.
  - ADS версии 1.7.1.b1 (для NiFi Server) и ADS версии 1.7.2.b1 (для NiFi Registry).

Организация взаимодействия кластеров по протоколу SSL

Для включения плагина Ranger необходимо, чтобы каждый кластер доверял сертификатам SSL другого.

Для этого:

1. Импортируйте сертификат **.crt* для каждого хоста кластера в *truststore* каждого хоста другого кластера.
   На каждом хосте, куда производится импорт сертификатов, выполните:

```
$ keytool -import -file /tmp/sov-test-1.ru-central1.internal.crt -keystore /tmp/truststore.jks -storepass bigdata -noprompt
```

2. На всех хостах выполните повторный импорт *truststore* в хранилище доверенных сертификатов операционной системы:

```
$ keytool -importkeystore -srckeystore /tmp/truststore.jks -destkeystore /etc/pki/java/cacerts -deststorepass changeit -srcstorepass bigdata -noprompt
```

3. На каждом хосте добавьте сертификаты в *ca-bundle.pem* с помощью следующих команд:

```
$ chmod 777 /etc/pki/ca-trust/extracted/pem/ca-bundle.pem
$ echo "##########################" >> /etc/pki/ca-trust/extracted/pem/ca-bundle.pem
$ cat /tmp/sov-test-1.ru-central1.internal.crt >> /etc/pki/ca-trust/extracted/pem/ca-bundle.pem
$ chmod 444 /etc/pki/ca-trust/extracted/pem/ca-bundle.pem
```

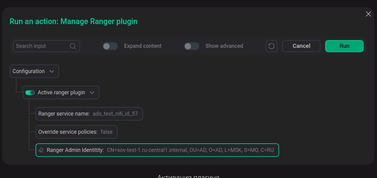
ПРИМЕЧАНИЕ

Если в кластере ADPS для включения SSL используются сертификаты Active Directory, в *truststore.jks* хостов кластера ADS необходимо импортировать корневой и клиентские сертификаты Active Directory.

Активация плагина Ranger NiFi

Для активации плагина Ranger NiFi:

1. Инициируйте активацию плагина Ranger NiFi. Для этого примените действие Manage Ranger plugin, нажав на иконку в столбце ***Actions*** сервиса NiFi.
2. В открывшемся окне ***Run an action: Manage Ranger plugin*** активируйте переключатель ***Active ranger plugin***. При этом:
   - Установите при необходимости название сервиса Ranger, который будет добавлен. Если сервис с таким именем уже существует, его можно пересоздать, активировав параметр ***Override service policies*** (`true` — параметр активен, `false` — неактивен), в таком случае старый сервис будет удален, а для нового сервиса будут заново сгенерированы политики. Если параметр ***Override service policies*** неактивен – после активации плагина осуществится подключение к существующему сервису Ranger.
   - Установите ***Ranger Admin Identitity*** – DN сертификата хоста ADPS, который был сформирован во время настройки SSL при помощи самоподписанных сертификатов и импортирован на хост (например, `CN=sov-test-1.ru-central1.internal, OU=AD, O=AD, L=MSK, S=MO, C=RU`). Этот DN Ranger будет использовать для коммуникации с NiFi.

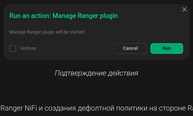


Активация плагина

ПРИМЕЧАНИЕ

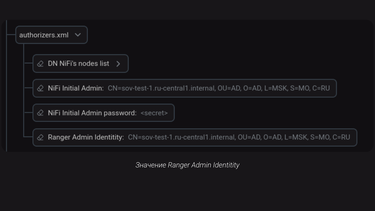
При первом включении плагина сервис с политиками будет создан (если еще не существует) вне зависимости от значения параметра ***Override service policies***.

3. Кликните ***Run*** и подтвердите действие в открывшемся окне:



Подтверждение действия

4. Дождитесь окончания процесса активации плагина Ranger NiFi и создания дефолтной политики на стороне Ranger. Проанализируйте и исправьте ошибки в случае их возникновения.
5. Проверьте, что в дереве конфигурационных параметров на странице конфигурации сервиса NiFi свойство ***Ranger Admin Identitity*** в группе ***authorizers.xml*** заполнилось значением DN.

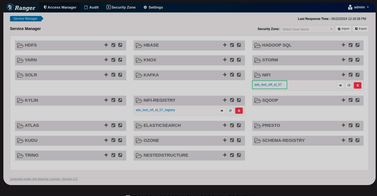


Значение Ranger Admin Identitity

Проверка активированного плагина Ranger NiFi

После авторизации в веб-интерфейсе Ranger в окне ***Service manager*** появятся сервисы для компонентов NiFi выбранного кластера ADS.

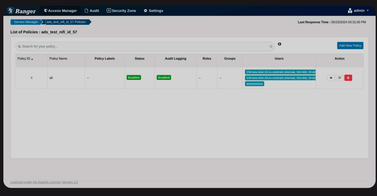
Для просмотра созданных политик для кластера нажмите на название сервиса Ranger для компонентов NiFi.



Переход к созданному сервису политик

При этом становится доступна политика, автоматически созданная для серверов кластера.

Для просмотра, редактирования или удаления политик используются кнопки в столбце ***Action***.



Созданная политика для компонента NiFi Server

Использование Ranger в NiFi

Denis Astahov

Содержание

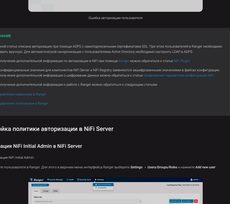
Настройка политики авторизации в NiFi Server
- Авторизация NiFi Initial Admin в NiFi Server
- Авторизация нового пользователя

Работа авторизованного пользователя в NiFi

Настройка политики авторизации в NiFi Registry
- Авторизация NiFi Initial Admin в NiFi Registry

Перед началом работы с Ranger для подключения к существующему серверу (в зависимости от значения параметров `ranger.service.*`) должен быть установлен и запущен сервис [Ranger Admin](#) и [NiFi Server](#), зависеть для чего пользователь, а так же для администратора NiFi - указанный в **NiFi Initial Admin** при настройке [Authorizers.xml](#) пользователь.

После настройки сервиса под NiFi Initial Admin появляется сообщение об отсутствии авторизованной политики для пользователя:



Ошибка авторизации пользователя

ПРИМЕЧАНИЕ

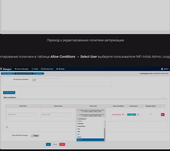
- В данной статье используется версия Ranger 2.0.0 с установленным сервисом NiFi. При этом пользователей в Ranger необходимо создавать вручную. Для автоматической синхронизации с пользовательским Active Directory необходимо настроить LDAP в AD/PS.
- Для получения дополнительной информации по настройке в NiFi. Как говорится [docs](#), можно обратиться к статье [NiFi Plugin](#).
- Все конфигурационные значения для настройки NiFi Server и NiFi Registry, записанные закодированными значениями в файлах конфигурации, можно получить с помощью [encrypt-config.sh](#) и [encrypt-config.cmd](#), зависеть для чего пользователю [Администрирование и настройка кластера NiFi](#).
- Для получения дополнительной информации о работе с Ranger можно обратиться к следующим статьям:
  - [Управление правилами в Ranger](#)
  - [Управление политиками в Ranger](#)

Настройка политики авторизации в NiFi Server

Авторизация NiFi Initial Admin в NiFi Server

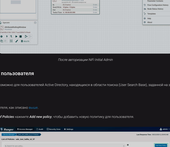
Для авторизации NiFi Initial Admin:

1. Создать пользователя в Ranger. Для этого в верхнем меню интерфейса Ranger выбрать **Settings** → **Users/Groups/Roles** и нажать **Add new user**



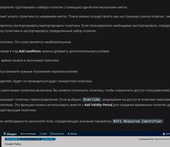
Переход к созданию пользователя в Ranger

2. В открывшемся окне заполнить необходимые данные для пользователя:
   - **User Name** – имя пользователя. Имя должно совпадать с именем пользователя в Base Active Directory. Значение может быть:
     - полный DN пользователя при условии значения конфигурационного параметра **Identity Strategy** группы **LDAP Login Identity Provider** = `USE_DN`
     - только имя пользователя при условии **Identity Strategy** = `USE_USERNAME`.
   - **New Password** – пароль пользователя
   - **Password Confirm** – подтверждение пароля пользователя
   - **First Name** – значение имя пользователя
   - **Last Name** – фамилия пользователя
   - **Email Address** – email адрес пользователя
   - **Select Role** – выбор роли пользователя: `Admin` `User` `Auditor`
   - **Group** – выбор группы/групп, к которым принадлежит пользователь.



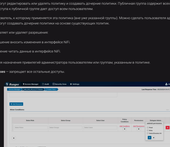
Создание пользователя в Ranger

3. Кликнуть **Save**
4. В окне **Service Manager** кликнуть на название сервиса в Ranger для NiFi.



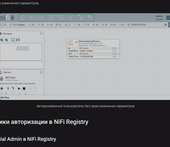
Переход к созданию сервису политик

5. В открывшемся окне **List of Policies** нажать кнопку **Add New Policy**.



Переход к редактированию политики авторизации

6. В открывшемся окне редактирования политики в таблице **Allow Conditions** → **Select User** выбрать пользователя NiFi Initial Admin, созданного ранее.

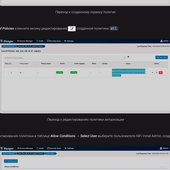


Добавление пользователя в политику доступа

7. Внизу страницы кликнуть **Save**.

В результате пользователь NiFi Initial Admin назначены полные права.

После настройки политики доступа в пользовательском интерфейсе NiFi Server появляется возможность авторизационной как NiFi Initial Admin. При этом в меню настроек отсутствует вкладка **Users** – настройка авторизации пользователей осуществляется только через Ranger.



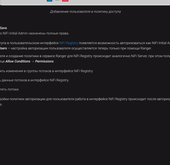
После авторизации NiFi Initial Admin

Авторизация нового пользователя

Авторизовать пользователя другого домена для пользователей Active Directory, находящихся в области поиска (User Search Base), зависеть от этого конфигурация параметров сервиса NiFi.

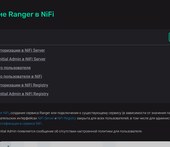
Для этого:

1. Создать нового пользователя, как описано [выше](#).
2. В открывшемся окне **List of Policies** нажать **Add new policy**, чтобы добавить новую политику для пользователя.



Создание политики авторизации

3. В открывшемся окне **Create Policy**:
   - заполнить поля в разделе **Policy Details**:
     - **Policy Name** – имя политики. Это имя не может быть добавлено для такого же сервиса в системе. Поле обязательно для заполнения.
     - **Policy Label** – вводит текстовое описание политики.
     - **Resource Identifiers** – позволяет ограничить доступ к ресурсам, указанным в политике.
     - Позволяет ограничивать группировать наборы политик с помощью одной или несколько записей.
     - Пользователь может искать политики по названию метки. Поиск можно осуществлять как по строкам полностью, так и по строкам отчета.
     - Поиск по ограничениям автоматически контролировать политики. Если пользователю необходимо экспортировать определенный набор политик, он может найти метку политики и экспортировать определенный набор политик.
     - **Description** – краткое описание. Это поле является необязательным.
     - **Policy Conditions** – начало в поле **Add conditions**, можно добавить дополнительные условия.
     - **Add Validity Period** – время начала и окончания политики.



Раздел Policy Details и Resource в окне Create Policy

   - заполнить поля в разделе **Allow Condition**:
     - **Select Role** – роль, которая применяется эта политика. Роль – набор разрешений. Роли предназначают собой более простой способ управления набором разрешений на основе определенного контроля доступа.
     - **Select Group** – группа пользователей, для которых применяется политика. Чтобы назначить группе пользователей, установите флажок **Delegate Admin**. Администраторы могут управлять/изменять или удалять политику с помощью этого пользователя. Публичная группа содержит всех пользователей, которому предоставляются разрешения публичной группе для этого конкретного ресурса.
     - **Select User** – пользователь, к которому применяется политика. Можно сделать пользователя администратором этой политики. Администраторы могут управлять, редактировать политику или полностью удалить пользователя.
     - **Permissions** – добавляет/ или удаляет разрешения:
       - `Write` – разрешение писать в конфигурацию NiFi.
       - `Read` – разрешение читать данные в конфигурации NiFi.
     - **Delegate Admin** – дает назначение привилегий администратора пользователям или группам, указанным в политике.
     - **Deny All Other Accesses** – запрещает все остальные доступы.

Раздел Allow Condition окна Create Policy

ПРИМЕЧАНИЕ

Условия выполняются в порядке, указанном в политике. Сначала применяются условия, находящиеся в начале списка, затем второе, третье и так далее.

4. Кликнуть **Add**.

Работа авторизованного пользователя в NiFi

После авторизации Ranger и настройки политики авторизации для пользователя в интерфейсе NiFi Server происходит после авторизации в соответствии с назначенной политикой доступа. Например, политика назначена только разрешение `Read`, панель инструментов неактивна и возможно только просмотр изменения кластера без изменения параметров.

Авторизованный пользователь без прав изменения параметров

Настройка политики авторизации в NiFi Registry

Авторизация NiFi Initial Admin в NiFi Registry

Для авторизации NiFi Initial Admin в NiFi Registry:

1. В окне **Service Manager** кликнуть на название сервиса Ranger для NiFi Registry.

Переход к созданию сервису политик

2. В открывшемся окне **List of Policies** кликнуть иконку редактирования **Add new policy**.

Переход к редактированию политики авторизации

3. В открывшемся окне редактирования политики в таблице **Allow Conditions** → **Select User** выбрать пользователя NiFi Initial Admin, созданного ранее.

Добавление пользователя в политику доступа

4. Внизу страницы кликнуть **Save**.

В результате пользователь NiFi Initial Admin назначены полные права.

После настройки Ranger и настройки политики авторизации для пользователя работа в интерфейсе NiFi Registry происходит после авторизации в соответствии с назначенной политикой доступа.

Работа с процессорами

Last update:

Содержание

Назад к содержанию

- Отдельный процессор
  - Создание процессора
  - Управление процессором
  - Подключение и отключение процессоров
- Группы процессов
  - Создание группы процессов
  - Управление группой процессов
  - Подключение и отключение группы процессов

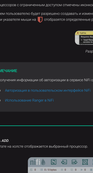
Работа с процессорами осуществляется в интерфейсе NiFi Server.

Отдельный процессор

Создание процессора

Создание процессора производится в следующей последовательности:

1. Перенести иконку Process на холст.
2. Выбрать нужный процессор в открывшемся диалоговом окне. При выборе можно воспользоваться тегами или обзором тегов.



Выбор процессора

Процессоры с ограниченным доступом

Виды процессоров с ограниченным доступом отмечены иконкой.

Прежде чем пользователи будут допущены к изменению процессоров с ограниченным доступом, им должен быть предоставлен доступ. При наведении указателя мыши на предоставляется следующее, которые требуется для процессоров с ограниченным доступом:

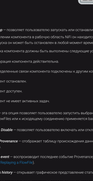
Разрешение для использования процессора

ПРИМЕЧАНИЕ

Для получения информации об авторизации в сервисе NiFi обратитесь к следующим статьям:

- Авторизация и управление доступом интерфейса NiFi
- Использование Ranger в NiFi

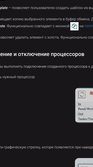
3. Кликнуть **ADD**.
В результате на холсте отобразится выбранный процессор.



Процессор на холсте

Управление процессором

Взаимодействие с процессором на холсте осуществляется при помощи перемещения контекстного меню, вызываемого правой кнопкой мыши.



Контекстное меню процессора

Параметры, доступные в контекстном меню, различаются в зависимости от назначенных пользователю разрешений. Следующие параметры обычно доступны, когда есть назначенные для работы с процессором:

- **Configure** – позволяет пользователю устанавливать или изменять настройки процессора. Функциональность совпадает с иконкой и с пунктом операций.
После создания процессор будет иметь отметки ошибок, где указаны не которые отображаются параметры, необходимые для работы процессора.

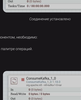
Необходимые параметры

- **Start** и **Stop** – позволяет пользователю запускать или останавливать процессор. Функциональность совпадает с иконкой и с пунктом операций.
При добавлении компонента в рабочую область он находится в состоянии остановки. Для перевода компонента в другое состояние его необходимо запустить. После запуска останавливать его можно в любой момент времени. В случае остановки компонент может находиться, запускать или отключить.
Для запуска компонента должны быть выполнены следующие условия:
  - Конфигурация компонента действительна.
  - Все определенные связи компонента подключены, в другом компоненте или настроены на автоматическое завершение.
  - Компонент остановлен.
  - Компонент доступен.
  - Компонент не имеет активных задач.
- **Run Once** – эта опция позволяет пользователю запустить выбранный процессор ровно один раз. Если процессор не может выполнить (например, нет входных файлов FlowFiles или в очередных соединениях недостаточно свободного места), процессор будет ожидать, пока условие не будет выполнено.
- **Enable** или **Disable** – позволяет пользователю включать или отключать процессор. Функциональность совпадает с иконками и с пунктом операций.
- **View Data Provenance** – отображает таблицу происхождения данных NiFi с информацией о событиях происхождения данных для FlowFiles, направленных через этот процессор.
- **View usage** – переводит пользователя к документации по использованию процессора.
- **View connections** | **Upstream** – позволяет пользователю видеть в "перспективе" с находящими соединениями, поступающими в процессор. Это особенно полезно, когда процессоры подключаются к другим группам процессов и выходят из них.
- **View connections** | **Downstream** – позволяет пользователю видеть в "перспективе" с находящими соединениями, которые выходят из процессора. Это особенно полезно, когда процессоры подключаются к другим группам процессов и выходят из них.
- **Center in view** – центрирует представление холста на данном процессоре.
- **Change color** – позволяет пользователю изменить цвет процессора. Это может упростить визуальное управление большим потоком. Функциональность совпадает с иконкой и с пунктом операций.
- **Create template** – позволяет пользователю создать шаблон из выбранного процессора. Функциональность совпадает с иконкой и с пунктом операций.
- **Copy** – позволяет копии выбранного элемента в буфер обмена. Данное элемент можно вставить в другое место на холсте, нажав правой кнопкой мыши на холсте и выбрав **Paste**. Функциональность совпадает с иконкой и с пунктом операций.
- **Delete** – позволяет удалить элемент с холста. Функциональность совпадает с иконкой и с пунктом операций.

Подключение и отключение процессоров

Для того, чтобы выполнить подключение создаваемого процессора к другому процессору или другому компоненту, необходимо выполнить:

1. Выделить нужный процессор.

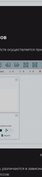


Выбор процессора

2. Перенести графическую стрелку, которая появляется при наведении на процессор в отступить над подключаемым компонентом (другим процессором).

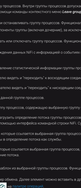
Подключение

3. В открывшейся конфигурации соединение настроить параметры соединения и кликнуть **ADD**.



Добавление соединения

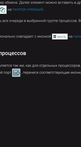
В результате устанавливается соединение между двумя компонентами.



Соединение установлено

Чтобы разорвать соединение процессора с другим компонентом, необходимо:

1. Остановить процессор при помощи иконки и пункта операций.
2. Выделить линию соединения.



Выделение линии соединения

3. Удалить линию соединения при помощи **DELETE** контекстного меню или иконки и пункта операций.

Группа процессов

Группы процессов можно использовать для логической организации набора компонентов, чтобы упростить понимание и обслуживание потока данных.

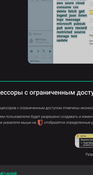
Создание группы процессов

Создание группы процессов производится в следующей последовательности:

1. Перенести иконку Process Group на холст.
2. Ввести наименование группы процессов в открывшемся диалоговом окне.

Ввод наименования группы процессов

3. Кликнуть **ADD**.
В результате на холсте отображается созданная группа процессов.



Группа процессов на холсте

Управление группой процессов

Взаимодействие с группой процессов на холсте осуществляется при помощи параметров контекстного меню, вызываемого правой кнопкой мыши.

Контекстное меню группы процессов

Параметры, доступные в контекстном меню, различаются в зависимости от назначенных пользователю разрешений. Следующие параметры обычно доступны, когда есть назначенные для работы с группой процессов:

- **Configure** – позволяет пользователю изменять имя, комментарий, конфигурацию группы процессов. Функциональность совпадает с иконкой и с пунктом операций.
- **Variables** – позволяет пользователю создавать или настраивать переменные в пользовательском интерфейсе NiFi.
- **Version** | **Start version control** – позволяет начать контроль версий.
- **Enter Group** – позволяет пользователю войти в группу процессов. Внутри группы процессов допускаются составить потоки как из отдельных процессоров, так и групп процессов. Войти из группы процессов можно при помощи контекстного меню **Leave group**.
- **Start** и **Stop** – позволяет пользователю запускать или останавливать группу процессов. Функциональность совпадает с иконкой и с пунктом операций.
При запуске группы процессов запускаются все процессоры, которые не отключены и не имеют недействительных конфигураций.
- **Enable** или **Disable** – позволяет пользователю включать или отключать группу процессов. Функциональность совпадает с иконкой и с пунктом операций.
- **View Data Provenance** – отображает таблицу происхождения данных NiFi с информацией о событиях происхождения данных для FlowFiles, направленных через группу процессов.
- **View status history** – открывает графическое представление статистической информации группы процессов с течением времени.
- **View connections** | **Upstream** – позволяет пользователю видеть в "перспективе" с находящими соединениями, поступающими в группу процессов.
- **View connections** | **Downstream** – позволяет пользователю видеть в "перспективе" с находящими соединениями, которые выходят из группы процессов.
- **Center in view** – центрирует представление холста на данной группе процессов.
- **Group** – позволяет пользователю создать новую группу процессов, содержащую выбранную группу процессов и любые другие компоненты, выбранные на холсте.
- **Download flow definition** – позволяет пользователю загрузить определение потока группы процессов в виде файла JSON. Файл можно использовать в качестве резервной копии или импортировать в рамках NiFi. Следующие опции доступны в контекстном меню, которое появляется при нажатии правой кнопкой мыши на холсте:
  - **Without external services** – служебы конечности, на которые ссылаются выбранные группы процессов, но которые находятся вне их области действия (например, службы в родительской группе), будут включены в определение потока.
  - **With external services** – служебы конечности, на которые ссылаются выбранные группы процессов, но не входящие в их область действия (например, службы в родительской группе), будут включены в определение потока.
- **Create template** – позволяет пользователю создать шаблон из выбранной группы процессов. Функциональность совпадает с иконкой и с пунктом операций.
- **Copy** – позволяет копии выбранного элемента в буфер обмена. Данное элемент можно вставить в другое место на холсте, нажав правой кнопкой мыши на холсте и выбрав **Paste**. Функциональность совпадает с иконкой и с пунктом операций.
- **Empty All Queues** – позволяет пользователю очистить все очереди в выбранной группе процессов. Все FlowFiles во всех соединениях, находящихся на момент запуска, будут удалены.
- **Delete** – позволяет удалить элемент с холста. Функциональность совпадает с иконкой и с пунктом операций.

Подключение и отключение группы процессов

Подключение и отключение группы процессов осуществляется так же, как для отдельных процессоров. При этом перед подключением группы процессов необходимо внутри группы развернуть входной порт выходной порт и перенести соответствующие входные и выходные порты на холст.

Настройка процессоров и групп процессов

Содержание

Назад к содержанию

Настройка процессоров
- Вкладка SETTINGS
- Вкладка SCHEDULING
- Вкладка PROPERTIES
- Вкладка RELATIONSHIPS
- Вкладка COMMENTS

Настройка групп процессов
- Вкладка GENERAL
- Вкладка CONTROLLER SERVICES

Настройка соединений
- Вкладка DETAILS
- Вкладка SETTINGS

Настройка процессоров

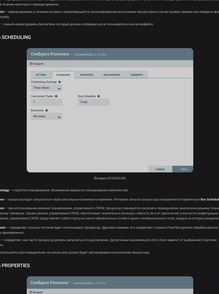
Чтобы настроить процессор, щелкните правой кнопкой мыши на процессор и выберите пункт **Configure**. В качестве меню доступен также контекстное меню, которое открывается при щелчке правой кнопкой мыши на процессор в рабочем поле. После щелчка на пункте процессора должно появиться окно **Configure**. Это окно состоит из нескольких вкладок: **SETTINGS**, **SCHEDULING**, **PROPERTIES**, **RELATIONSHIPS** и **COMMENTS**.

ВНИМАНИЕ

- После запуска процессора изменения не применяются. Чтобы применить изменения, сначала остановите процессор, внесите изменения, а затем снова запустите процессор. Все текущие задачи должны завершиться, прежде чем изменения вступят в силу.
- Текущие изменения сохраняются не поддерживается в автоматическом режиме. Следующие символы могут быть недопустимыми: `<>:"/\|?*` и символы, которые являются управляющими последовательностями.

`[0x00], [0x01], [0x02], [0x03], [0x04], [0x05], [0x06], [0x07], [0x08], [0x09], [0x0A], [0x0B], [0x0C], [0x0D], [0x0E], [0x0F], [0x10], [0x11], [0x12], [0x13], [0x14], [0x15], [0x16], [0x17], [0x18], [0x19], [0x1A], [0x1B], [0x1C], [0x1D], [0x1E], [0x1F]`

Вкладка SETTINGS



Вкладка SETTINGS

Name – имя процессора. Имя процессора по умолчанию совпадает с типом процессора. Рядом с именем процессора отмечен флажок **Enabled** указывающий включен ли процессор.

Id – уникальный идентификатор процессора.

Type – тип процессора.

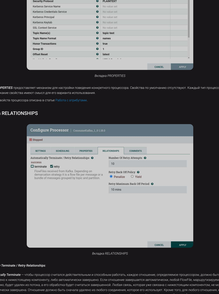
Bundle – пакет NiFi, в котором находится процессор.

Yield Duration – период времени, в течение которого процессор приостанавливается при получении сигнала "отдача" от внешнего оборудования или внутренней логики. Это предотвращает постоянные попытки работы, которые не будут успешными.

Penalty Duration – период времени, в течение которого процессор приостанавливается при получении сигнала о ручном наказании потока данных. Это предотвращает повторные попытки обработки данных, которые, скорее всего, будут неудачными.

Bulletin Level – самый низкий уровень бюллетеня, который должен регистрироваться в пользовательском интерфейсе.

Вкладка SCHEDULING



Вкладка SCHEDULING

Scheduling Strategy – стратегия планирования. Возможные варианты планирования описаны ниже.

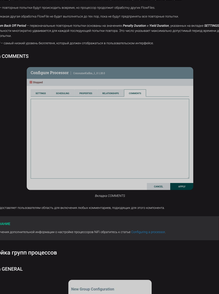
- **Timer driven** – процессор будет запускаться через регулярные промежутки времени. Интервал запуска процессора определяется параметром **Run Schedule**.
- **CRON driven** – при использовании режима планирования, определяемого CRON, процессор планируется работать в определенные моменты времени в соответствии с хорошо определенным и гибким форматом планирования, унаследованным от утилиты cron. Для получения дополнительной информации см. документацию Quartz.

Concurrent Tasks – определяет сколько потоков будет использовать процессор. Другими словами, это определяет сколько потоков может вызвать процессор одновременно.

Run Schedule – определяет, как часто процессор должен запускаться по расписанию. Допустимые значения для этого поля зависят от выбранной стратегии планирования.

Execution – используется для определения, на каком узле (узлах) будет запланировано выполнение процессора.

Вкладка PROPERTIES

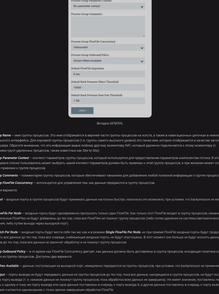


Вкладка PROPERTIES

Вкладка **PROPERTIES** предоставляет элементы для настройки поведения конкретного процессора. Свойства по умолчанию отсутствуют. Каждый тип процессора должен определить, какие свойства ему доступны, и какие значения по умолчанию.

Настройка свойств процессора описана в статье Работа с атрибутами.

Вкладка RELATIONSHIPS



Вкладка RELATIONSHIPS

Automatically Terminate / Retry Relationships

- **Automatically Terminate** – чтобы гарантировать сделать дальнейшую обработку невозможной, каждое отношение, определенное процессором, должно быть либо автоматически завершено, либо должно быть определено, что оно должно повторяться. Например, чтобы NiFi не выбрасывал информацию в отношение success, которое не было определено, процессор должен быть настроен для автоматического завершения отношения success. Если отношение автоматически завершается, все FlowFiles, которые передаются в это отношение, будут удалены из потока данных, как только процессор завершит свою сессию. Если отношение не будет автоматически завершено, процессор будет блокироваться, если отношение станет слишком большим. Для получения дополнительной информации см. документацию Backpressure.
- **Automatically Retry** – в качестве альтернативы завершению, можно настроить FlowFile, маршрутизируемый в данное отношение, для повторной попытки.

Number of Retry Attempts – для отношений, настроенных на повторную попытку, это число попыток, которые должен сделать FlowFile перед маршрутизацией в альтернативное отношение.

Retry Maximum Back Off Period – максимальное значение случайных попыток отложить на основе **Penalty Duration** указанного на вкладке **SETTINGS**. Время ожидания будет увеличиваться экспоненциально с каждой последующей попыткой, но не будет превышать значение, указанное в этом поле.

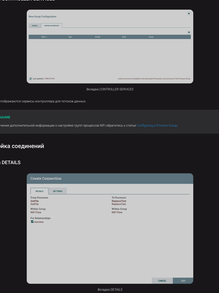
Retry Back Off Policy – когда FlowFile неоднократно повторно пытается, есть две стратегии, которые можно использовать:

- **Penalize** – повторяет попытку через заданный промежуток времени.
- **Yield** – снижает скорость работы всего потока данных, если это целесообразно. Чаще всего будет использоваться политика Penalize.

Penalty Duration и **Yield Duration**, указанные на вкладке **SETTINGS**, будут использоваться для этого поведения.

Bulletin Level – самый низкий уровень бюллетеня, который должен регистрироваться в пользовательском интерфейсе.

Вкладка COMMENTS



Вкладка COMMENTS

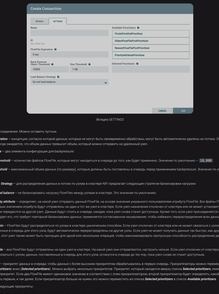
Вторая закрытая вкладка настройки процессора – это вкладка комментариев. Комментарии, предоставленные для этого компонента.

ПРИМЕЧАНИЕ

Для получения дополнительной информации о настройке процессора NiFi обратитесь в статью Configuring a processor.

Настройка групп процессов

Вкладка GENERAL



Вкладка GENERAL

Process Group Name – имя группы процессов. Это имя отображается на холсте. Имена не должны должны содержать точки и должны быть уникальными на данном уровне в потоке данных. Имена могут содержать буквы, цифры, пробелы, дефисы, подчеркивания и точки. Имена не должны начинаться или заканчиваться пробелом.

Process Group Parameter Context – позволяет параметризовать имена и значения свойств для текущего контекста параметров потока. В этом контексте параметров можно определить имена и значения свойств, которые можно использовать в группе процессов, а при желании может создать новый контекст параметров. Для получения дополнительной информации см. документацию Parameter Contexts.

Process Group Comments – комментарии группы процессов, которые обеспечивают некоторое для описания группы процессов.

Process Group FlowFile Concurrency – определяет, как должны обрабатываться FlowFiles, поступающие в группу процессов. Доступны три варианта:

- **Unbounded** – позволяет запускать в группе процессов при доступных потоках. Это может привести к тому, что при большом количестве FlowFiles в очереди, они будут обработаны не в том порядке, в котором они были получены.
- **Single FlowFile Per Node** – позволяет группе быстро запланировать процессоры только один FlowFile. Как только этот FlowFile входит в группу процессов, внешние процессоры FlowFiles не будут запланированы для группы процессов, пока FlowFile не покинет группу процессов. Если FlowFile разделен на несколько FlowFiles, они также будут обработаны до того, как будет запущен следующий FlowFile.
- **Single Batch Per Node** – аналогичен выбору выше, но все, что в результате этого FlowFile **Per Node**, из того времени FlowFile входящие потоки будут ограничены, чтобы содержать только данные, которые были получены от этого FlowFile. Как только FlowFile покинет, все FlowFiles будут помещены в группу процессов.

Process Group Outbound Policy – это то же самое что FlowFile Concurrency данную для данного потока группы процессов, исходящие потоки ограничивают данные, которые будут выходить из группы процессов.

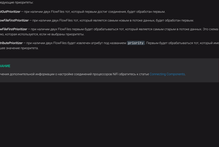
- **Stream When Available** – данные, поступающие из исходной точки, немедленно передаются из группы процессов, при условии, что быстрота не ограничена.
- **Batch Output** – группа выходов будет передавать данные из группы процессов до тех пор, пока после данные, находящиеся в группе процессов, не будут поставлены в очередь с пустой очередью. В результате данные, которые будут переданы из группы процессов, будут переданы в виде пакета. Если группа процессов содержит несколько выходных портов, данные будут переданы из группы процессов, как только данные, находящиеся в группе процессов, будут поставлены в очередь с пустой очередью. Данные будут переданы в виде пакета. Если группа процессов содержит несколько выходных портов, данные будут переданы из группы процессов, как только данные, находящиеся в группе процессов, будут поставлены в очередь с пустой очередью.

Default Settings for Connections – используется для FlowFile Concurrency.

Default Back Pressure Object Threshold – порог данных, используемый по умолчанию.

Default Back Pressure Data Size Threshold – порог данных, используемый по умолчанию.

Вкладка CONTROLLER SERVICES



Вкладка CONTROLLER SERVICES

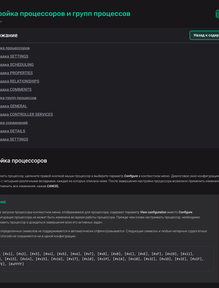
На вкладке отображаются сервисы контроллера для потока данных.

ПРИМЕЧАНИЕ

Для получения дополнительной информации о настройке групп процессов NiFi обратитесь в статью Configuring a Process Group.

Настройка соединений

Вкладка DETAILS



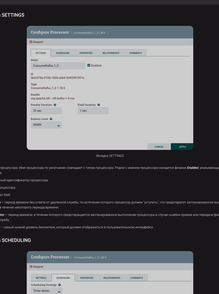
Вкладка DETAILS

На вкладке **DETAILS** предоставлена информация об исходном и целевом компонентах, включая имя компонента, тип компонента и группу процессов, в которой находится компонент.

For Relationships – предоставляет возможность выбрать, какие отношения должны быть доступны для этого соединения. Если доступно только одно отношение, все выбранные отношения.

Если с одного и того же исходного компонента несколько соединений, любой FlowFile, маршрутизируемый к этому отношению. Будет автоматически "клонирован", и его копия будет помещена в каждое из этих соединений.

Вкладка SETTINGS



Вкладка SETTINGS

Name – имя соединения. Можно оставить пустым.

FlowFile Expiration – указывает, сколько времени данные, которые могут быть неправильно обработаны, могут оставаться данных на потоке. Это полезно, если данные имеют ограниченную ценность, и вы хотите, чтобы данные, которые не удалось обработать.

Back Pressure – для влияния конфигурации для ограничения:

- **Object Threshold** – максимальное количество FlowFile, которое может находиться в очереди. Значение по умолчанию: **10_000**
- **Size Threshold** – максимальный общий размер (по размеру), которые могут быть поставлены в очередь перед применением backpressure. Значение по умолчанию: **1 GB**

Load Balance Strategy – для настройки балансировки нагрузки между узлами в кластере NiFi определяет следующие стратегии балансировки нагрузки:

- **Do not Load Balance** – данные не будут балансироваться. FlowFiles всегда остаются на том узле, где они были обработаны.
- **Partition by attribute** – определяет, на каком узле определенный FlowFile должен обрабатываться основываясь на значении атрибута FlowFile. Все FlowFiles с одинаковым значением для данного атрибута будут отправлены на один и тот же узел. Если FlowFile не имеет указанного атрибута, он будет считаться имеющим пустое значение для этого атрибута. Если большое количество различных значений для указанного атрибута, это может привести к неравномерному распределению данных.
- **Round robin** – FlowFiles будут распределены по узлам в кластере циклическим способом. Если один отключен и не может принять с узлом, данные, предназначенные для этого узла, будут распределены между оставшимися узлами. Когда узел снова может принять данные, он получит данные в следующем цикле. Этот режим распределения данных приведет к равномерному распределению данных, но потребует больше сетевого трафика, чем другие стратегии.
- **Single node** – все FlowFile будут отправлены на один узел в кластере. На какой узел они отправляются, неясно. Если узел отключен и снова вне не может принять данные, данные, предназначенные для этого узла, будут распределены между оставшимися узлами. Когда узел снова может принять данные, он получит данные в следующем цикле.

Prioritizers – приоритет данных в очереди. Если данные должны обрабатываться в каком-то порядке, Предоставляется список приоритетов, которые будут использоваться для определения порядка, в котором данные будут извлекаться из очереди. Доступны следующие приоритеты:

- **FirstInFirstOutPrioritizer** – при запросе данных FlowFile, тот, который является самым ранним в потоке данных, будет обработан первым.
- **NewestFlowFileFirstPrioritizer** – при запросе данных FlowFile, тот, который является самым новым в потоке данных, будет обработан первым.
- **OldestFlowFileFirstPrioritizer** – при запросе данных FlowFile, тот, который является самым старым в потоке данных. Это самое по умолчанию для всех очередей, и это не должно изменяться.
- **PriorityAttributePrioritizer** – при запросе данных FlowFile будет извлечен атрибут под названием **priority**. Первым будет обрабатываться тот, который имеет наименьшее значение приоритета.

ПРИМЕЧАНИЕ

Для получения дополнительной информации о настройке соединений процессоров NiFi обратитесь в статью Connecting Components.

Работа с атрибутами

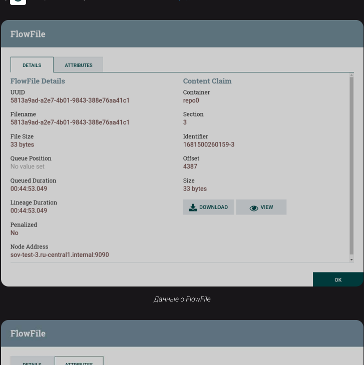
Ольга Семин

Содержание

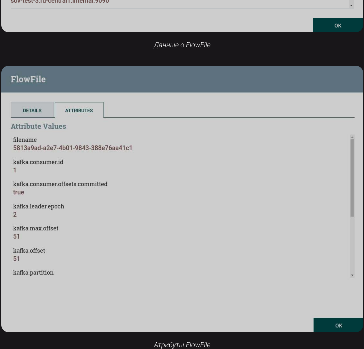
- Атрибуты FlowFile
- Свойства процессора
- Параметры
  - Создание контекста параметров
  - Создание параметров в контексте параметров
  - Работа с контекстом параметров в интерфейсе NiFi server
  - Проверка зависимых компонентов
- Переменные
  - Создание переменной
  - Использование переменных в свойствах процессора с применением языка выражений
  - Проверка зависимых компонентов
- Параметры с использованием языка выражений

Атрибуты FlowFile

Все данные в Apache NiFi представлены абстракцией, называемой FlowFile. FlowFile состоит из двух основных частей: содержимого и атрибутов. Информация о FlowFile доступна при нажатии на иконку на странице *List queue* контекстного меню очереди потока.



Данные о FlowFile



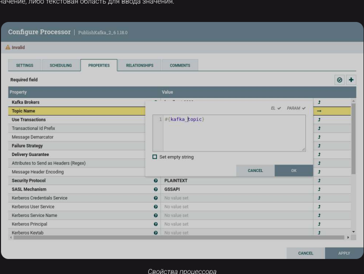
Атрибуты FlowFile

Полный список основных атрибутов приведен в разделе FlowFile руководства разработчика NiFi.

Атрибуты FlowFile могут быть извлечены и использованы в качестве свойств процессоров при настройке потоков в интерфейсе NiFi.

Свойства процессора

Во время настройки процессора на вкладке **PROPERTIES** отображаются свойства процессора. В столбце **Property** указано наименование параметра. В столбце **Value** указывается значение параметра. В зависимости от значений, разрешенных для свойства, в столбце **Value** пользователю предоставляется либо раскрывающийся список, из которого можно выбрать значение, либо текстовая область для ввода значения.

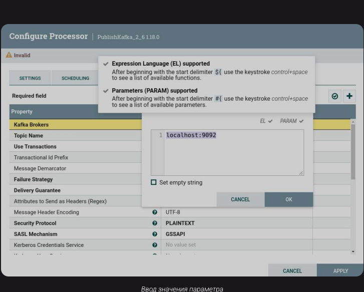


Свойства процессора

Для параметров с текстовой областью для ввода значений существуют следующие формы ввода значения:

- напрямую;
- с использованием параметров, определяемых при помощи контекста параметров;
- с использованием переменных, задаваемых при помощи языка выражений Expression Language;
- с использованием параметров вместе с языком выражений (подробно описано в Parameters and Expression Language).

В верхней части окна для ввода значения указано, поддерживаются ли Expression Language (EL) и Parameters (PARAM) для ввода значения конкретного свойства данного процессора.



Ввод значения параметра

Параметры имеют следующие преимущества перед переменными:

- поддержка конфиденциальных значений;
- более детальный контроль над политиками доступа;
- свойства, ссылающиеся на параметры, проверяются на соответствие замененному значению, в отличие от большинства свойств, ссылающихся на переменные с помощью языка выражений.

Параметры

Значения свойств в потоке, включая конфиденциальные свойства, можно параметризовать с помощью параметров. Параметры создаются в окне **Parameter Contexts** глобального меню интерфейса NiFi Server. Контексты параметров глобально определяются для каждого экземпляра NiFi. К контекстам параметров можно применять политики доступа, чтобы определить каким пользователям могут их создавать.

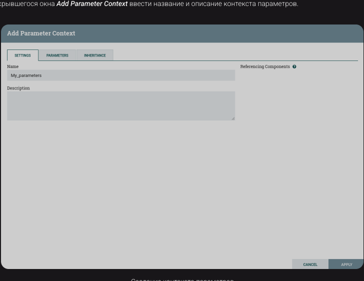
Любое свойство можно настроить для ссылки на параметр со следующими условиями:

- Конфиденциальное свойство может ссылаться только на конфиденциальный параметр.
- Неконфиденциальное свойство может ссылаться только на неконфиденциальный параметр.
- Свойства, которые ссылаются на службы контроллера, не могут использовать параметры.
- На параметры нельзя ссылаться в задачах создания отчетов или в службах контроллера управления.

Создание контекста параметров

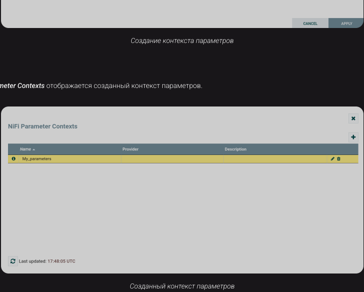
Чтобы создать контекст параметров, необходимо:

1. Через глобальное меню интерфейса NiFi Server открыть окно **Parameter Contexts**.
2. Нажать + для создания контекста параметров.
3. На вкладке **SETTINGS** открывшегося окна **Add Parameter Context** ввести название и описание контекста параметров.



Создание контекста параметров

4. Нажать **APPLY**.
   В результате в окне **Parameter Contexts** отображается созданный контекст параметров.



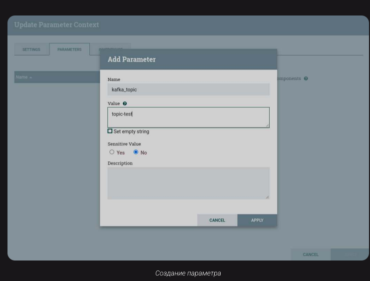
Созданный контекст параметров

Создание параметров в контексте параметров

> **ПРИМЕЧАНИЕ**
> Допускается как создавать параметры при создании контекста параметров, так и добавлять параметры уже в созданный контекст параметров.

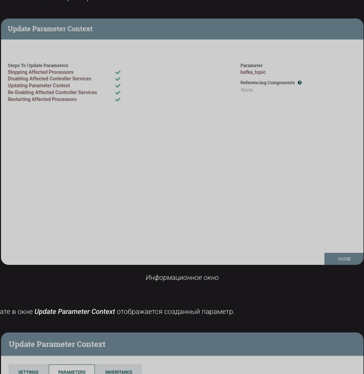
Чтобы создать новый параметр в контексте параметров, необходимо:

1. Открыть для редактирования созданный контекст параметров, кликнув в строке контекста параметров на иконку редактирования.
2. На вкладке **PARAMETERS** нажать + для создания параметра.
3. В открывшемся окне **Add Parameter** заполнить поля для параметра:
   - **Name** — имя, которое используется для обозначения параметра. Допускаются только буквенно-цифровые символы, дефисы, символы подчеркивания, точки и пробелы.
   - **Value** — значение, которое будет использоваться при ссылке на параметр. Если параметр использует язык выражений, важно отметить, что язык выражений будет оцениваться в контексте компонента, который ссылается на параметр. Дополнительные сведения об использовании языка выражений в параметрах приведены ниже.
   - **Set empty string** — отметьте этот флажок для установки значения параметра в пустую строку. Не отмечен по умолчанию. Если флажок установлен, но значение указано, флажок игнорируется.
   - **Sensitive Value** — установите значение `Yes`, если значение параметра следует считать конфиденциальным. Если параметр конфиденциален, значение параметра не будет отображаться в пользовательском интерфейсе после применения. Значение по умолчанию — `No`. На конфиденциальные параметры могут ссылаться только конфиденциальные свойства, а на неконфиденциальные параметры — неконфиденциальные свойства. После создания параметра значение его конфиденциальности нельзя изменить.
   - **Description** — описание параметра. Это поле является необязательным.



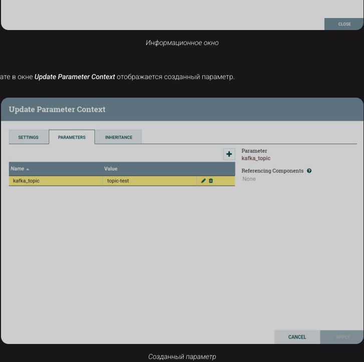
Создание параметра

4. Нажать **APPLY**.
   В результате открывается информационное окно по обновлению или созданию параметра. Для проверки всех компонентов, которые ссылаются на добавленные или измененные параметры, выполняются следующие операции: остановка или перезапуск зависимых процессоров, отключение или повторное включение зависимых служб контроллера, обновление контекста параметров.



Информационное окно

5. Нажать **CLOSE**. В результате в окне **Update Parameter Context** отображается созданный параметр.

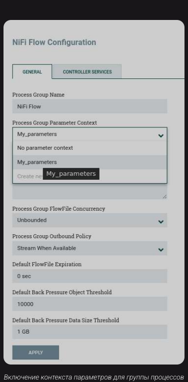


Созданный параметр

Работа с контекстом параметров в интерфейсе NiFi server

Для того, чтобы распространить созданный контекст параметров на группу процессов, необходимо выполнить:

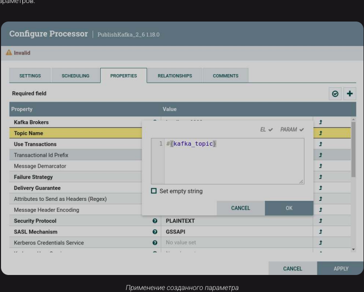
1. На свободном поле правой кнопкой мыши вызвать контекстное меню и выбрать команду **Configure**.
2. В открывшемся окне **NiFi Flow Configuration** на вкладке **GENERAL** для параметра **Process Group Parameter Context** во всплывающем списке выбрать необходимый контекст параметров.
3. Нажать **APPLY**.



Включение контекста параметров для группы процессов

Для того, чтобы применить созданный параметр из выбранного контекста параметров для конкретного процессора, необходимо выполнить:

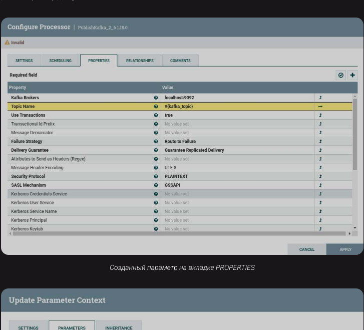
1. Для выделенного процессора правой кнопкой мыши вызвать контекстное меню и выбрать команду **Configure**.
2. На вкладке **PROPERTIES** выбрать нужный параметр и в открывшемся окне ввести значение параметра в виде `#{name}`, где `name` — имя, введенное при создании параметра в контексте параметров.



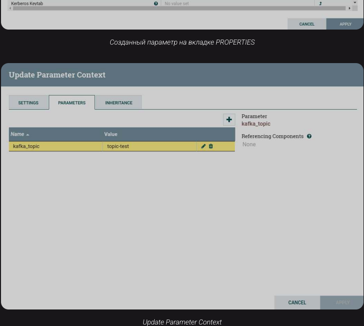
Применение созданного параметра

3. Нажать **OK**.

В случае успешного применения параметра в последнем столбце назначенного свойства появляется иконка , при клике на которую происходит переход на страницу **Update Parameter Context**, где данный параметр доступен к изменению.

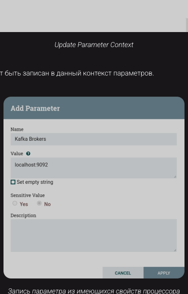


Созданный параметр на вкладке PROPERTIES



Update Parameter Context

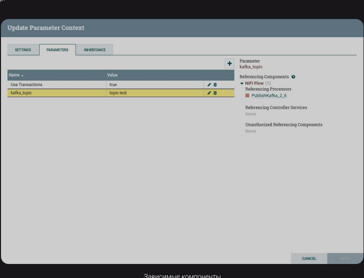
При клике на иконку каждый параметр также может быть записан в данный контекст параметров.



Запись параметра из имеющихся свойств процессора

Проверка зависимых компонентов

На вкладке **PARAMETERS** окна **Parameter Contexts** для выбранного параметра в строке **Referencing Components** отображаются наименования всех процессоров, в которых применяется данный параметр.



Зависимые компоненты

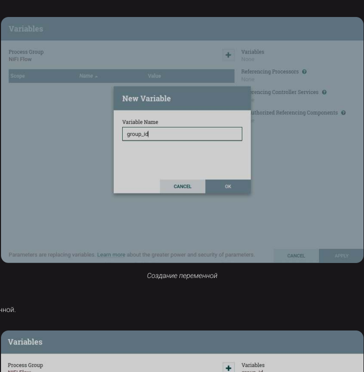
Переменные

Переменные создаются и настраиваются в пользовательском интерфейсе NiFi. Их можно использовать в любой области, поддерживающей Expression Language. Переменные нельзя использовать для конфиденциальных свойств. NiFi автоматически подбирает новые или измененные переменные. Переменные определяются на уровне группы процессов, поэтому политики доступа для просмотра и изменения переменных являются производными от политик доступа группы процессов.

Создание переменной

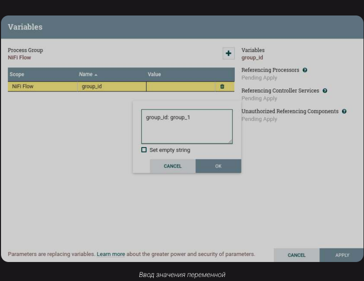
Чтобы создать переменную, необходимо:

1. На свободном поле правой кнопкой мыши вызвать контекстное меню и выбрать команду **Variables**.
2. В открывшемся окне нажать + для создания переменной.
3. Ввести наименование переменной.
4. Кликнуть **OK**.



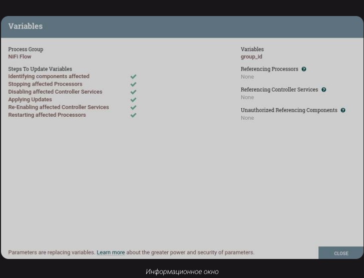
Создание переменной

5. Ввести значение переменной.



Ввод значения переменной

6. Кликнуть **OK**.
7. В окне **Variables** нажать **APPLY**, в результате открывается информационное окно по обновлению или созданию переменной.

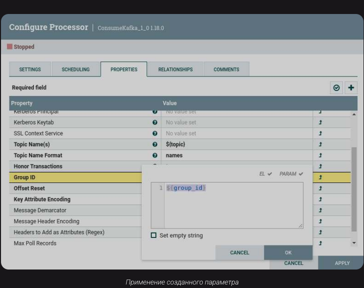


Информационное окно

Использование переменных в свойствах процессора с применением языка выражений

Для того, чтобы применить созданную переменную для конкретного процессора, необходимо выполнить:

1. Для выделенного процессора правой кнопкой мыши вызвать контекстное меню и выбрать команду **Configure**.
2. На вкладке **PROPERTIES** выбрать нужный параметр и в открывшемся окне ввести значение параметра в виде `${name}`, где `name` — имя, введенное при создании переменной.



Применение созданного параметра

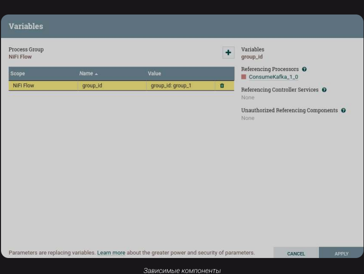
> **Функции Expression Language**
>
> При помощи различных функций Expression Language существует возможность манипулировать значением переменных. Некоторые функции обеспечивают манипуляции со строками (текстом), например, функция `toUpper`. Другие, такие как функции `equals` и `match`, обеспечивают функциональность сравнения. Также существуют функции для управления датами и временем и для выполнения математических операций.
>
> Пользуясь необходимой структурой выражения, возможно, например, вернуть версию имени файла в верхнем регистре, вызвав функцию `toUpper` `${filename:toUpper()}`. В этом случае ссылаются на атрибут имени файла, а затем манипулируют этим значением с помощью функции `toUpper`.
>
> Вызов функции состоит из 5 элементов:
>
> - Разделитель вызова функции `:`.
> - Имя функции, в данном случае `toUpper`.
> - Открывающая скобка `(`.
> - Аргументы функции. Необходимые аргументы зависят от того, какая функция вызывается. В этом примере используется функция `toUpper`, у которой нет аргументов, поэтому этот элемент опущен.
> - Закрывающая скобка `)` указывает на конец вызова функции.

3. После ввода функции для переменной нажать **OK** в окне ввода значения свойства и **APPLY** на вкладке **PROPERTIES** в окне **Configure** процессора.

> **ПРИМЕЧАНИЕ**
> Переменные в дочерней группе процессов переопределяют значения в родительской группе. В частности, если переменная задана в группе root, а также получает иное значение внутри группы процессов, то в таком случае компоненты внутри группы процессов используют значение, определенное непосредственно в самой группе процессов.

Проверка зависимых компонентов

В окне **Variables** для выбранной переменной в строке **Referencing Components** отображаются наименования всех процессоров, в которых применяется данный параметр.



Зависимые компоненты

> **ВНИМАНИЕ**
> При создании настраиваемых свойств убедитесь, что каждое настраиваемое свойство содержит отдельное значение свойства, чтобы оно не перекрывалось существующими свойствами среды, системными свойствами или атрибутами FlowFile.

Параметры с использованием языка выражений

При добавлении параметра, использующего язык выражений, важно понимать контекст, в котором будет оцениваться язык выражений. Выражение всегда оценивается в контексте службы процессора или контроллера, которая ссылается на параметр. Возьмем, к примеру, сценарий, в котором у параметра с именем `Time` значением является `${now()}`. Результатом выражения является вызов для определения системного времени при его оценке. При добавлении параметра `Time` в качестве свойства системное время оценивается не при добавлении параметра, а при оценке выражения службой процессора или контроллера. То есть, если процессор имеет свойство, значение которого установлено на `#{Time}`, он будет работать точно так же, как если бы значение свойства было установлено на `${now()}`. Каждый раз, когда на свойство ссылаются, оно создает другую отметку времени.

Кроме того, некоторые свойства не позволяют использовать язык выражений, в то время как другие разрешают использовать язык выражений, но не оценивают выражение по атрибутам FlowFile. Чтобы понять, как это работает, рассмотрим параметр с именем File, значение которого равно `${filename}`. Затем рассмотрим три разных свойства, каждое из которых имеет свою область языка выражений и FlowFile с именем файла test.txt. Если для каждого из этих свойств задано значение `#{File}`, результирующее значение показано в следующей таблице.

Настроенное значение свойства	Область действия Expression Language	Эффективное значение свойства	Примечание
#{File}	Атрибуты FlowFile	test.txt	Имя файла разрешается просмотром атрибута имени файла
	Только реестр переменных	Пустая строка	Атрибуты FlowFile не входят в область действия, и предполагается, что в реестре переменных нет переменной с именем `filename`
	Нет	${filename}	Буквальный текст `${filename}` не будет оцениваться

Работа с шаблонами

Ольга Семина

Содержание

Назад к содержанию

- Управление шаблонами
  - Создание шаблона
  - Экспорт шаблона в виде XML-файла
  - Импорт шаблона
- Использование шаблона

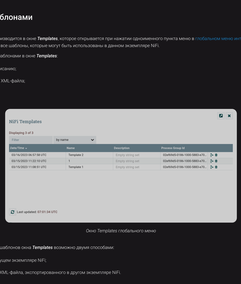
Шаблоны в интерфейсе NiFi Server используются при необходимости создания нескольких типовых одинаковых секций в большой схеме потока. При помощи шаблона возможно объединение в группы основных компонентов, созданных при помощи панели инструментов.

Управление шаблонами

Управление шаблонами производится в окне **Templates**, которое открывается при нажатии одноименного пункта меню в глобальном меню интерфейса NiFi Server. В данном окне представлены все шаблоны, которые могут быть использованы в данном экземпляре NiFi.

Возможные действия над шаблонами в окне **Templates**:

- фильтр по имени или описанию;
- экспорт шаблона в виде XML-файла;
- удаление шаблона.



Окно Templates (глобального) меню

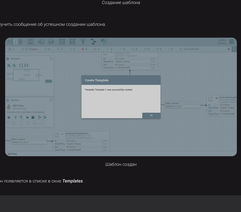
Добавить шаблон в список шаблонов окна **Templates** возможно двумя способами:

- создание шаблона в текущем экземпляре NiFi;
- импорт шаблона в виде XML-файла, экспортированного в другом экземпляре NiFi.

Создание шаблона

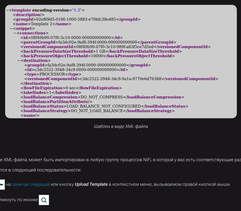
Создание шаблона производится в следующей последовательности:

1. На холсте выбрать компоненты, которые должны быть частью шаблона. Несколько компонентов можно выбрать, удерживая клавишу SHIFT.



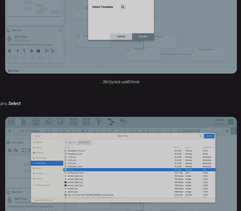
Выделенные компоненты

2. Кликнуть по иконке в панели операций или кнопку **Create Template** в контекстном меню, вызываемом правой кнопкой мыши.
3. В открывшемся окне ввести наименование шаблона и, при необходимости, его описание.



Создание шаблона

4. Кликнуть **CREATE** и получить сообщение об успешном создании шаблона.



Шаблон создан

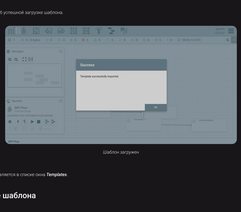
После создания шаблон появляется в списке в окне **Templates**.

ПРИМЕЧАНИЕ

Если запустить создание шаблона, кликнув на свободном поле холста (внутри группы процессов) без выбора компонентов, шаблон будет содержать все компоненты текущей группы процессов (присутствующие на данном холсте).

Экспорт шаблона в виде XML-файла

После появления шаблона в списке окна **Templates** существует возможность сохранить данный шаблон в виде XML файла. Для этого необходимо в строке с выбранным шаблоном кликнуть по иконке . После загрузки файл *template_name.xml* доступен к просмотру в XML представлении.



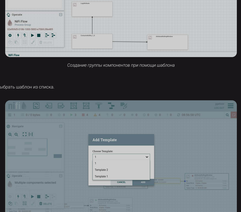
Шаблон в виде XML-файла

Импорт шаблона

Шаблон, сохраненный в виде XML-файла, может быть импортирован в любую группу процессов NiFi, в которой у вас есть соответствующие разрешения.

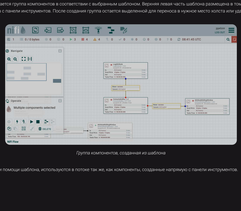
Импорт шаблона производится в следующей последовательности:

1. Кликнуть по иконке в панели операций или кнопку **Upload Template** в контекстном меню, вызываемом правой кнопкой мыши.
2. В открывшемся окне кликнуть по иконке .



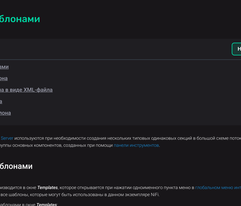
Загрузка шаблона

3. Выбрать шаблон и нажать **Select**.



Выбор шаблона

4. Убедиться, что выбран правильный шаблон и кликнуть **UPLOAD**.



Загрузка шаблона

5. Получить сообщение об успешной загрузке шаблона.

Шаблон загружен

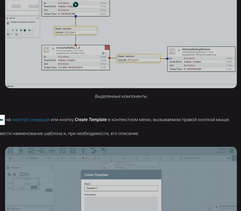
После импорта шаблон появляется в списке окна **Templates**.

Использование шаблона

После создания или импорта шаблон готов к использованию на холсте.

Создание группы компонентов на холсте при помощи шаблона производится в следующей последовательности:

1. Перенести иконку с панели инструментов на холст.



Создание группы компонентов при помощи шаблона

2. В открывшемся окне выбрать шаблон из списка.

Выбор шаблона из списка

3. Кликнуть **ADD**.

В результате на холсте создается группа компонентов в соответствии с выбранным шаблоном. Верхняя левая часть шаблона размещена в том же месте, куда была перенесена иконка шаблона с панели инструментов. После создания группа остается выделенной для перехода в нужное место холста или удаления при необходимости.

Группа компонентов, созданная из шаблона

Компоненты, созданные при помощи шаблона, используются в потоке так же, как компоненты, созданные напрямую с панели инструментов.

Встроенный мониторинг в интерфейсе NiFi

Ольга Семме

Назад к содержанию

Содержание

Строка состояния
Сводка данных о работе компонентов системы
- Данные о работе компонентов
- Работа с данными
- Данные о работе системы

Счетчики событий
Бюллетени
Процессор
Группа процессов
Удаленная группа процессов
Очередь

В данной статье описано, как в пользовательском интерфейсе NiFi представлена информация о потоке и его компонентах для отслеживания его работоспособности и состояния.

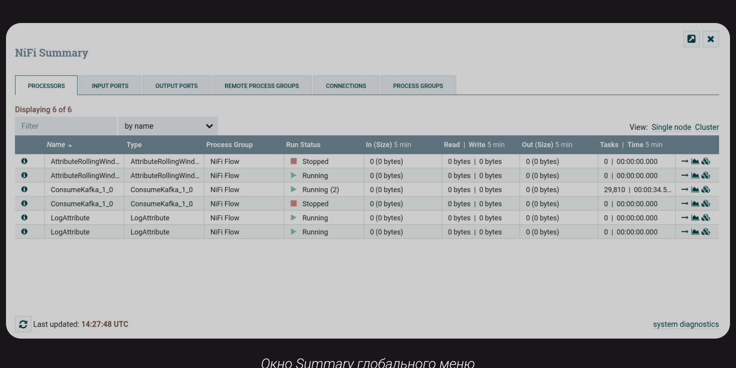
Строка состояния

Информация об общем состоянии системы представлена в строке состояния интерфейса NiFi Server.

Сводка данных о работе компонентов системы

Сводка данных о работе всех компонентов системы представлена в окне ***Summary***, которое открывается при нажатии одноименного пункта меню в глобальном меню интерфейса NiFi Server.

Сводка предоставляет информацию обо всех основных компонентах, созданных при помощи панели инструментов, в табличном формате.

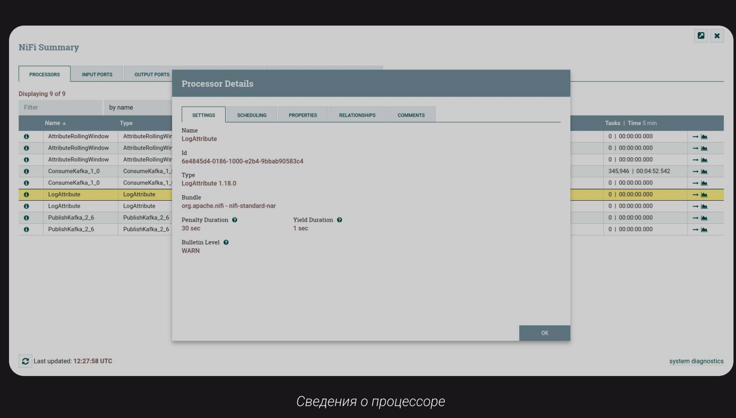


Окно Summary глобального меню

Данные о работе компонентов

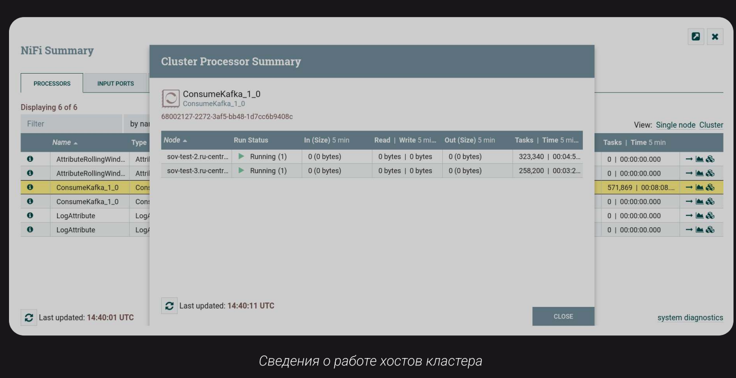
Информацию о выбранном компоненте можно получить, кликнув по соответствующей иконке:

- — предоставляет основные сведения о компоненте. Данные окна повторяют данные окна ***Configure*** контекстного меню, вызываемого правой кнопкой мыши.



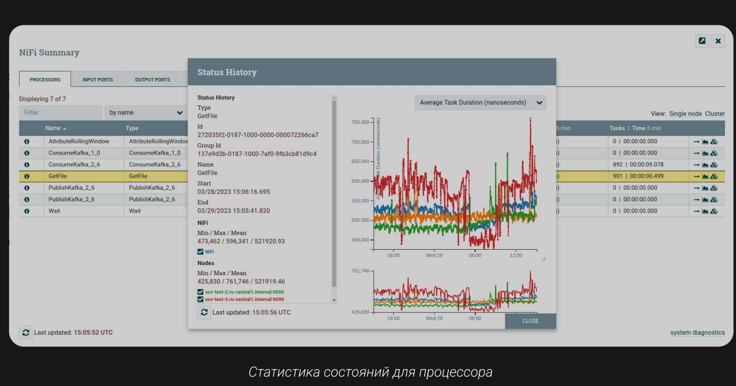
Сведения о процессоре

- — предоставляет основные данные по работе каждого хоста кластера. Данная иконка доступна только для кластера, состоящего из нескольких хостов.



Сведения о работе хостов кластера

- — отображает статистику состояний для выбранного компонента.



Статистика состояний для процессора

В левой части представлена следующая информация:

- ***Type*** — тип компонента.
- ***Id*** — идентификатор компонента, для которого отображается статистика.
- ***Group Id*** — идентификатор группы процессов, в которой находится компонент.
- ***Name*** — имя компонента, для которого отображается статистика.
- ***Start*** — самое раннее время, показанное на графике.
- ***End*** — последнее время, показанное на графике.
- ***Min/Max/Mean*** — минимальное, максимальное и среднее (среднее арифметическое или среднее) значения. Эти значения основаны только на выбранном диапазоне времени. Каждый узел отображается своим цветом.

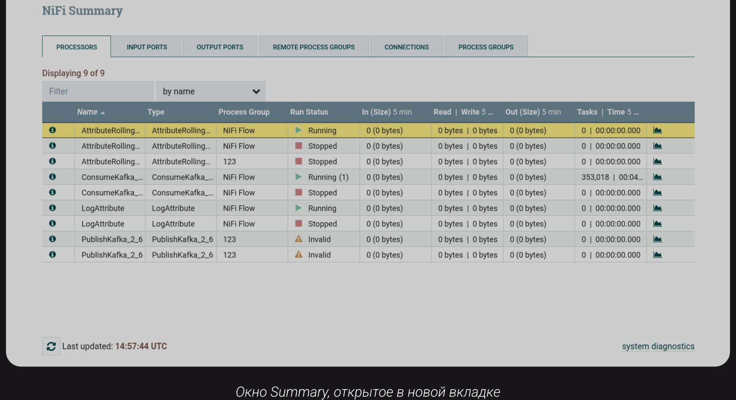
В правой части представлены:

- Раскрывающийся список различных типов метрик, отображаемых на графиках ниже.
- Увеличенный верхний график, обеспечивающий более легкое чтение информации. В правом нижнем углу этого графика находится небольшой маркер, который можно перетаскивать, чтобы изменить размер графика.
- Нижний график, предоставляющий возможность выбора временного диапазона для отображения на верхнем графике.

Работа с данными

Для навигации и упрощения пользования данными в окне ***Summary*** используются следующие элементы:

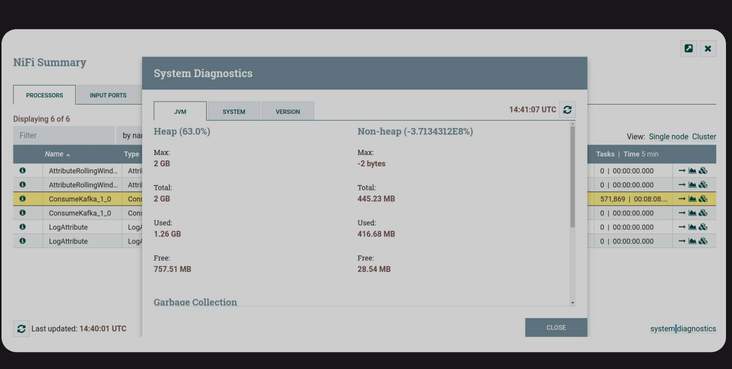
- Фильтр в верхней части таблицы позволяет пользователям фильтровать содержимое сводной таблицы, вводя все или часть некоторых критериев, таких как тип процессора или имя процессора.
- — позволяет обновить отображаемую информацию, не закрывая диалоговое окно и не открывая его снова. Информация на странице не обновляется автоматически.
- — закрывает страницу ***Summary*** и переносит пользователя непосредственно к компоненту на холсте NiFi. Это может изменить группу процессов, в которой в данный момент находится пользователь.
- — приводит к открытию всего диалогового окна ***Summary*** в новой вкладке или окне браузера. В открывшемся окне доступны к просмотру только данные, вызываемые при помощи иконки .



Окно Summary, открытое в новой вкладке

Данные о работе системы

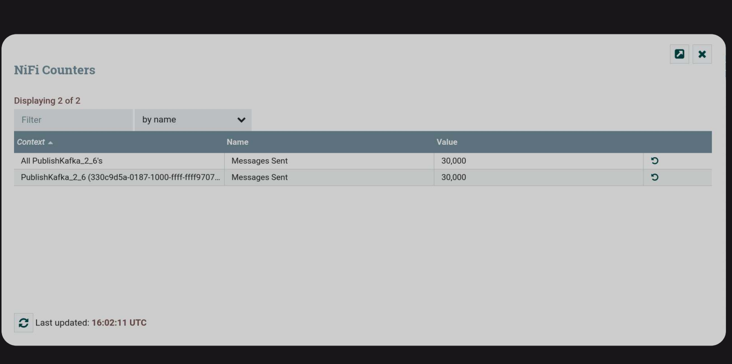
Для получения информации об использовании системных ресурсов необходимо кликнуть строку ***system diagnostics*** в правом нижнем углу окна ***Summary***. В открывшемся окне представлена сводная информация о системе.



Диагностика системы

Счетчики событий

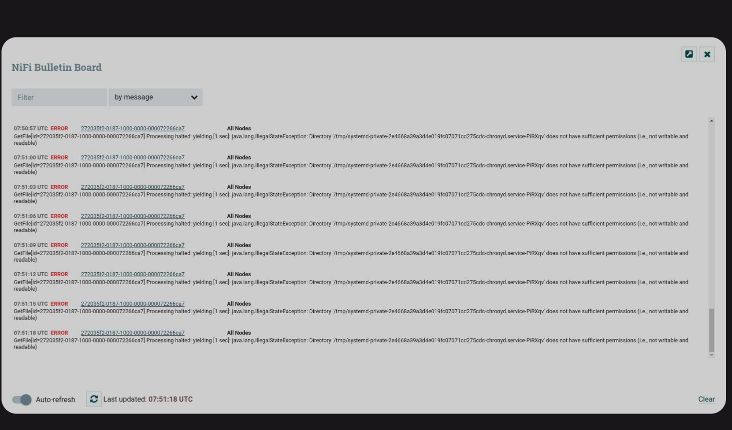
Счетчики событий по каждому компоненту представлены в окне ***Counters***, которое открывается при нажатии одноименного пункта меню в глобальном меню интерфейса NiFi Server.



Счетчики событий

Бюллетени

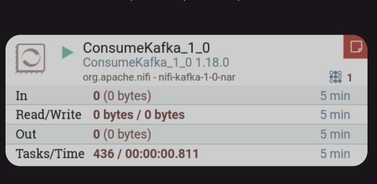
Бюллетени произошедших событий представлены в окне ***Bulletin Board***, которое открывается при нажатии одноименного пункта меню в глобальном меню интерфейса NiFi Server.



Бюллетени

Процессор

Графическое отображение процессора на холсте включает в себя данные о процессоре и его работе.



Процессор

Ниже описаны информационные элементы на примере представленного процессора:

- ***ConsumeKafka_1_0*** — определяемое пользователем имя процессора. По умолчанию имя процессора совпадает с типом процессора. Имя процессора можно изменить во время настройки процессора на вкладке ***SETTINGS***.
- ***ConsumeKafka_1_0 1.18.0*** — тип процессора. Данный процессор предназначен для приема сообщений от Apache Kafka, созданных для API Kafka 1.0 Consumer.
- ***org.apache.nifi - nifi-kafka-1-0-nar*** — название бандла процессора.
- — индикатор бюллетеня, обозначающий, что для данного процессора создан бюллетень в результате произошедшего события. Уровень бюллетеня можно изменить во время настройки процессора на вкладке ***SETTINGS***. Когда иконка присутствует, при наведении мыши на значок появится всплывающая подсказка, объясняющая сообщение, предоставленное процессором, а также уровень бюллетеня. Если экземпляр NiFi сгруппирован, он также покажет узел, выпустивший бюллетень. Бюллетени автоматически истекают через пять минут.
- Индикатор состояния, показывающий текущее состояние процессора. Возможны следующие индикаторы:
  - — процессор в данный момент работает.
  - — процессор действителен и включен, но не работает.
  - — процессор включен, но в настоящее время недействителен и не может быть запущен. При наведении курсора на этот значок появится всплывающая подсказка, указывающая, почему процессор недействителен.
  - — процессор не работает и не может быть запущен, пока он не будет включен. Этот статус не указывает, действителен ли процессор.
- — количество задач, которые в данный момент выполняет процессор. Ограничение количества задач настраивается при помощи параметра ***Concurrent Tasks*** во время настройки процессора на вкладке ***SCHEDULING***.
- 5-минутная статистика:
  - ***In*** — объем данных, которые процессор извлек из очередей своих входящих подключений. Это значение представлено в следующем формате: `<количество> (<размер>)`, где `<количество>` — количество FlowFile, которые были извлечены из очередей, а `<размер>` — общий размер содержимого этих FlowFiles, включая единицу измерения (например, `15 (20.1 MB)`).
  - ***Read/Write*** — общий размер содержимого FlowFile, прочитанного процессором с диска и записанного на диск.
  - ***Out*** — объем данных, которые процессор передал своим исходящим соединениям. Сюда не входят файлы FlowFiles, которые Процессор удаляет сам, или файлы FlowFiles, которые перенаправляются на автоматически завершаемые соединения. Это значение представлено в следующем формате: `<количество> (<размер>)`, где `<количество>` — количество FlowFile, которые были извлечены из очередей, а `<размер>` — общий размер содержимого этих FlowFiles, включая единицу измерения (например, `15 (20.1 MB)`).
  - ***Tasks/Time*** — количество раз, когда процессор запускался для запуска за последние 5 минут, и количество времени, затраченное на выполнение этих задач. Формат времени: `<часы>:<минуты>:<секунды>`.

Группа процессов

Графическое отображение группы процессов на холсте включает в себя данные о ее работе.

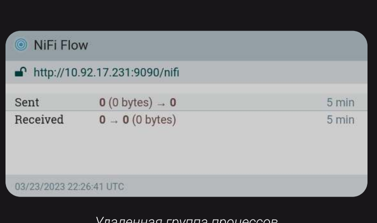
Группа процессов

Ниже описаны информационные элементы на примере представленной группы процессов:

- ***New group*** — имя, определяемое пользователем во время создания группы процессов. Имя группы процессов можно изменить во время настройки группы процессов на вкладке ***GENERAL***.
- — индикатор, обозначающий, что для для дочернего компонента группы процессов выдан бюллетень.
- — количество задач, которые в данный момент выполняют компоненты этой группы процессов.
- 5-минутная статистика:
  - ***Queued*** — количество файлов FlowFiles, находящихся в настоящее время в очереди в группе процессов. Это значение представлено в следующем формате: `<количество> (<размер>)`, где `<количество>` — количество FlowFile, которые были извлечены из очередей, а `<размер>` — общий размер содержимого этих FlowFiles, включая единицу измерения (например, `15 (20.1 MB)`).
  - ***In*** — количество файлов FlowFiles, которые были переданы в группу процессов через все входные порты за последние 5 минут. Это значение представлено в следующем формате: `<количество> (<размер>) → <порты>`, где `<количество>` — количество FlowFile, которые были извлечены из очередей, `<размер>` — общий размер содержимого этих FlowFiles, включая единицу измерения, а `<порты>` — количество входных портов.
  - ***Read/Write*** — общий размер содержимого FlowFile, которое компоненты в группе процессов считывали с диска и записывали на диск.
  - ***Out*** — количество файлов FlowFiles, которые были переданы из группы процессов через ее порты вывода за последние 5 минут. Это значение представлено в следующем формате: `<порты> → <количество> (<размер>)`, где `<порты>` — количество выходных портов, `<количество>` — количество FlowFile, которые были извлечены из очередей, `<размер>` — общий размер содержимого этих FlowFiles, включая единицу измерения.
- Счетчики состояний для компонентов в соответствии со строкой состояния в интерфейсе NiFi Server.

Удаленная группа процессов

Графическое отображение удаленной группы процессов на холсте включает в себя данные о ее работе.



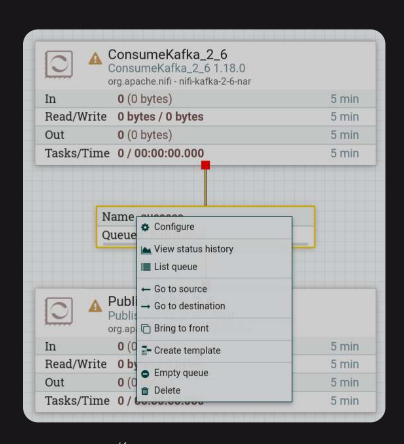
Удаленная группа процессов

Ниже описаны информационные элементы на примере представленной группы процессов:

- Статус передачи, указывающий, включена ли в данный момент передача данных между текущим и удаленным экземплярами NiFi. Возможны следующие показатели:
  - — передача активна, какой-либо из входных или выходных портов в настоящее время настроен на передачу;
  - — передача неактивна, если все входные и выходные порты, которые в настоящее время подключены, остановились.
- ***NiFi Flow*** — имя экземпляра NiFi, о котором сообщил удаленный экземпляр.
- ***http://10.92.17.231:9090/nifi*** — URL-адрес удаленного экземпляра, на который указывает группа удаленных процессов. Этот URL-адрес вводится при добавлении группы удаленных процессов на холст, и его нельзя изменить.
- индикатор безопасности, указывающий, защищена ли связь с удаленным экземпляром NiFi:
  - — связь защищена, этот экземпляр NiFi не сможет взаимодействовать с удаленным экземпляром, пока администратор удаленного экземпляра не предоставит доступ.
  - — связь не защищена.
- 5-минутная статистика:
  - ***Sent*** — количество файлов FlowFiles, которые были отправлены. Это значение представлено в следующем формате: `<количество> (<размер>) → <порты>`, где `<количество>` — количество FlowFile, которые были извлечены из очередей, `<размер>` — общий размер содержимого этих FlowFiles, включая единицу измерения, а `<порты>` — количество входных портов.
  - ***Received*** — это количество файлов FlowFiles, которые были получены. Это значение представлено в следующем формате: `<порты> → <количество> (<размер>)`, где `<порты>` — количество выходных портов, `<количество>` — количество FlowFile, которые были извлечены из очередей, `<размер>` — общий размер содержимого этих FlowFiles, включая единицу измерения.
- Время последнего обновления удаленного потока.

Очередь

Существует возможность просмотра диагностики очереди через контекстное меню.

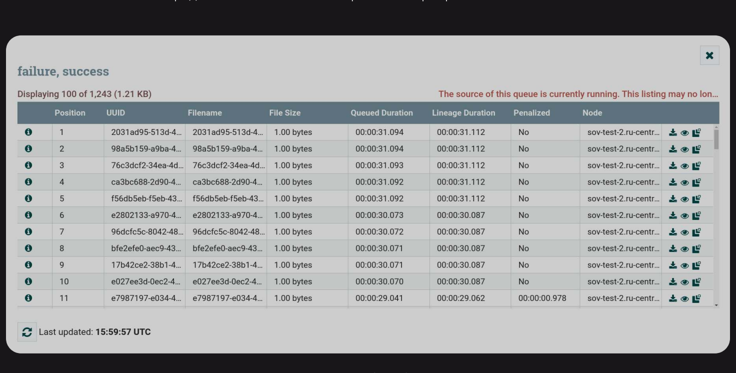


Контекстное меню очереди

View status history — отображает статистику состояний для очереди.

Статистика состояний для очереди

List queue — вернет 100 файлов FlowFiles в активной очереди в соответствии с настроенным встроенным приоритетом.



Список FlowFiles

ПРИМЕЧАНИЕ

Для получения дополнительной информации о мониторинге в пользовательском интерфейсе Apache NiFi обратитесь к статье Monitoring of DataFlow.

Использование языка выражений NiFi

Tim Kellogg

Содержание

Назад к содержанию

- Структура языка выражений
- Иерархия языка выражений
- Редактор языка выражений
- Типы данных для языка выражений
- Использование языка выражений при работе с процессорами
  - Пример ссылки на переменную
  - Пример ссылки на атрибут FlowFile
  - Пример значения параметра в контексте параметров

Язык выражений NiFi позволяет управлять значениями атрибутов FlowFile при настройке свойств процессоров.

Язык выражений также может применяться:

- при использовании ссылок на переменную;
- как значение параметра, созданного в контексте параметров.

Структура языка выражений

Язык выражений NiFi начинается с начального разделителя `${` и заканчивается конечным разделителем `}`. Между начальным и конечным разделителем находится само выражение. В самой простейшей форме выражение может состоять только из имени атрибута. Например, `${days.name}` вернет значение атрибута `days.name`.

Для получения значения атрибута, имя которого содержит специальные символы, используйте функцию `getAttribute()`.

Вы можете выполнить цепочку вызовов функции. Например, `${days.name:replaceAll('.*','2')}` в этом случае мы обращаемся к атрибуту `days.name`, а затем заменяем весь исходный атрибут на `2`.

Каждое значение атрибута в NiFi является строкой. Однако некоторые функции, как ветвление работает `ifElse()`, ожидают, что значение будет `boolean`.

Структура вызова функции

Вызов функции состоит из четырех частей:

- Разделитель вызова функции `${`.
- Имя функции, например `replace`.
- Открывающая скобка `(`.
- Аргументы функции. Необходимые аргументы зависят от того, какая функция вызывается. В этом примере используются аргументы:
  - `'.'` текст, который должен быть заменен.
  - `'2'` текст замены.
- Закрывающая скобка `)` указывает на конец вызова функции.

Не любой атрибут FlowFile можно ссылаться с помощью языка выражений. Однако, если имя атрибута содержит "специальный символ", имя атрибута необходимо экранировать, чтобы его можно было использовать в качестве "специального символа". `$`, `{`, `}`, `(`, `)`, `:`, `,` и `"` считаются "специальными символами". Чтобы использовать специальный символ, просто поместите `'` (одинарную кавычку) вокруг строки, в которой он находится. Если строка уже содержит `'`, экранируйте его, используя `''`. Кроме того, часто считается "специальным символом" все, что не является буквой или числом. Если вы не уверены, что символы считаются специальными, просто заключите в одинарные кавычки.

Также доступны функции `anyAttribute()` и `allAttributes()`. Они функции вызываются для каждого значения переменной, которая содержит функцию. Например, `${anyAttribute('').isEmpty()}` будет возвращать `true`, если любой из атрибутов пуст.

Иерархия языка выражений

При использовании языка выражений вы можете использовать несколько функций и операторов, которые могут быть объединены в иерархию.

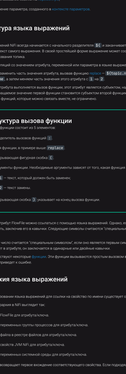
Текущая иерархия в NiFi выглядит так:

1. Поиск FlowFile по атрибуту.
2. Поиск переменных среды процесса для атрибута/значения.
3. Поиск файла в реестре файлов для атрибута/значения.
4. Поиск свойства в JVM NiFi для атрибута/значения.

NiFi ищет в вышеуказанной первое вхождение соответствующего свойства. Если подходящее не найдено, возвращается `null`.

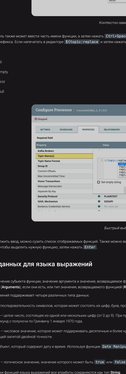
Редактор языка выражений

Настройка значения свойства в более удобном для использования редакторе языка выражений. Чтобы использовать, нажмите `Ctrl` + `Space` в редакторе. Это вызовет редактор кода, который предоставляет автозаполнение, подсветку синтаксиса и многое другое. Редактор также предоставляет встроенную документацию для каждой доступной функции.



Выделение ошибки

Редактор также предоставляет контекстно-зависимую справку. Вы можете выбрать одну из доступных функций, чтобы увидеть ее документацию, или просто поместить курсор в любое место внутри функции, чтобы увидеть документацию для нее. Для активации этой функции необходимо нажать `Ctrl`+`Space` из редактора.



Контекстно-зависимая справка

Пользователь также может вместо часть имени функции, а затем нажать `Ctrl`+`Space`, чтобы увидеть все функции, которые можно использовать в которые начинаются с того же префикса. Если начинается с вводом `${hasAttribute`, и затем нажать `Ctrl`+`Space` появится исчерпывающий список из списка функций:

- replace
- replaceAll
- replaceEmpty
- replaceFirst
- replaceNull

Быстрый выбор функции

Если предусмотреть ввод, можно сузить список функций. Например, можно выбрать одну из функций из списка, введя какое-нибудь или используя клавиши со стрелками, чтобы выбрать нужную, затем нажать `Enter`.

Типы данных для языка выражений

Каждая функция будет возвращать какой-либо результат. Каждая функция имеет определенный тип данных. Тип субъекта функции (**Subject Type**), тип аргумента (**Arguments**), если они есть, или тип возвращаемого результата (**Return Type**), указаны в описании каждой из функций.

Язык выражений поддерживает несколько типов данных для данных:

- **String** — последовательность символов, которые хранятся как строка.
- **Number** — целое число, состоящее из одной или нескольких цифр (от 0 до 9). При преобразовании в число из типов данных Date при представлении как количество миллисекунд с января 1 января 1970 года.
- **Decimal** — число, которое может содержать дробную часть, включая десятичную точку. Поддерживается до 15 цифр. Бит-в-бит IEEE 754-2 с поддержкой двойной точности.
- **Date** — объект, который содержит дату и время. Используя функцию `Date: format(String)`, этот тип данных может преобразовать в/из строк и чисел.
- **Boolean** — логическое значение, значения которого могут быть `true` или `false`.

После одного функций языка выражения все атрибуты определяются как тип **String**.

Использование языка выражений при работе с процессорами

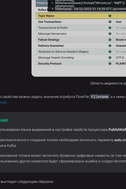
Язык выражений широко используется в интерфейсе NiFi. Среди для настройки свойств процессора. Не все свойства процессора поддерживают язык выражений. Информацию о том, поддерживается ли данное свойство процессора, можно узнать, изучив описание свойства в верхней части для для каждой значений или щелкнув пиктограмму на вкладке **PROPERTIES**.

Также при наличии на вкладке **PROPERTIES** с изменяемыми свойствами вверху, какая область выделяется для языка выражений поддерживается:

- **NONE** — язык выражений не поддерживается для этого свойства.
- **VARIABLE_REGISTRY** — язык выражений для использования в рамках переменных. При этом также применяется выполняется переменных.
  - Переменные определяются на уровне группы процесса, а затем, унаследуют до группы процессов более высокого уровня до корневой группы процессов.
  - Переменные, определенные в файлах конфигурации свойств через `nifi.variable.registry.properties` в файле url-адресов.
  - Переменные среды, определенные на уровне JVM и свойствах системы.
- **FLOWFILE_ATTRIBUTES** — для языка выражений включены атрибуты с каждого отдельного потока потока, а также переменных.

Пример ссылки на переменную

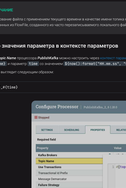
Для процессора **PublishKafka** при настройке свойства **Kafka Brokers** включает поддержку переменных.



Область выделена для языка выражений

Создав переменную `kafka.broker`, ее можно использовать `kafka.broker.internal:9092`. Значение переменной можно использовать, установив при помощи языка выражений `kafka.broker.internal:9092`. Также `${kafka.broker}:9092` и `${kafka.broker:append(':9092')}` и т.д., они эквивалентны. Выражение для замены `${kafka.broker:replaceAll('[^:]+(:.+)?','kafka.broker.internal$1')}` и т.д., они эквивалентны. Выражение для замены `${kafka.broker:replaceAll('[^:]+(:.+)?','kafka.broker.internal$1')}` при помощи языка выражений.

```
${kafka.broker:replaceAll('.*','2')}
```



Указание нескольких значений с данной части текста

ПРИМЕЧАНИЕ

Замена символов может быть использована при большом количестве брокеров Kafka с именами хостов, отличающихся несколькими символами. При создании новых топиков потребуется, используя имена хостов для каждого брокера, можно использовать одно выражение для замены имен брокеров на имена брокеров. Это может быть использовано для легкого изменения брокеров Kafka, которые используются Kafka-брокеров с помощью одного изменения.

Пример ссылки на атрибут FlowFile

Далее, настраивая следующее свойство **Topic Name** для процессора **PublishKafka**, можно увидеть, что данное свойство разрешает использовать атрибуты FlowFile.

Область выделена для языка выражений

Для данного свойства можно задать значение атрибута FlowFile `filename` и к нему присоединить значений текущего времени при помощи функции `now()` применимого формата `format`.

ВНИМАНИЕ

При использовании языка выражений в настройке свойств процессора **PublishKafka** необходимо учитывать следующее:

- Для автоматического создания топиков необходимо включить **auto.create.topics.enable** в группе **server.properties** при конфигурировании сервера Kafka.
- Применяемые точки может включать буквенно-цифровые символы (в том числе в кодировке ASCII), а также символы `.`, `_` и `-`. При использовании для символов будет сформирован ошибок и создан топик/очередь.

Выражение выглядит следующим образом:

```
${filename}_${now():format("MM.dd.yy", "GMT+1")}
```


Назначение файла с применением текущего времени в качестве имени топика

Тогда, если значение атрибута `filename` в данном FlowFile — `dataflow`, процессор **PublishKafka** публикует данные в топик `dataflow_12.24.14` — `12.24.14` — время публикации топика в формате "mm.dd.yy", а часовая пояса "GMT+1".

ПРИМЕЧАНИЕ

Назначение файла с применением текущего времени в качестве имени топика может быть использовано при необходимости публикации в топики Kafka данных из FlowFile, создаваемого на часовых поясах с различными временами.

Пример значения параметра в контексте параметров

Свойство **Topic Name** процессора **PublishKafka** можно использовать параметр из контекста параметров. Если в контексте параметров создать параметр `Topic` со значением `${filename}` и параметр `time` со значением `${now():format("MM.dd.yy", "GMT+1")}`, можно применить созданные параметры в качестве свойства.

Выражение выглядит следующим образом:

```
#{topic}_#{time}
```


Назначение файла с применением текущего времени в качестве имени топика

Тогда, если значение атрибута `filename` в данном FlowFile — `dataflow`, процессор **PublishKafka** публикует данные в топик `dataflow_12.24.14` — как и в предыдущем примере.

Шифрование параметров конфигурации NiFi

Ольга Семме

Содержание

Назад к содержанию

Общие данные о шифровании

Шифрование новых данных при использовании сервиса

Изменение пароля

Сведения о шифровании конфиденциальных данных в файлах конфигурации NiFi

Общие данные о шифровании

После установки сервиса NiFi все конфиденциальные значения в открытом тексте для компонентов NiFi Server и NiFi Registry заменяются зашифрованными значениями в следующих файлах конфигурации:

- *nifi.properties*
- *authorizers.xml*
- *login-identity-providers.xml*
- *nifi-registry.properties*

Шифрование выполняется автоматически средствами ADCM с использованием инструмента шифрования конфигурации encrypt-config, который считывает из файлов чувствительные к открытому тексту значения конфигурации, запрашивает **корневой ключ** (**root key**) и шифрует каждое значение (заменяет простые значения защищенным значением в том же файле).

Используемый алгоритм шифрования по умолчанию — 128/256-битный AES/GCM.

Корневой ключ генерируется из пароля, указанного в конфигурационном параметре ***Nifi config encryption password***, и записывается в файл *bootstrap.conf* командой шифрования для сохранения и предоставления конфигурации схемы защиты при необходимости.

Для файла *bootstrap.conf* должны быть предоставлены разрешения на чтение только тому пользователю, который может его выполнить (пользователь `nifi`, пользователь из группы `nifi` или `root`).

Шифрование новых данных при использовании сервиса

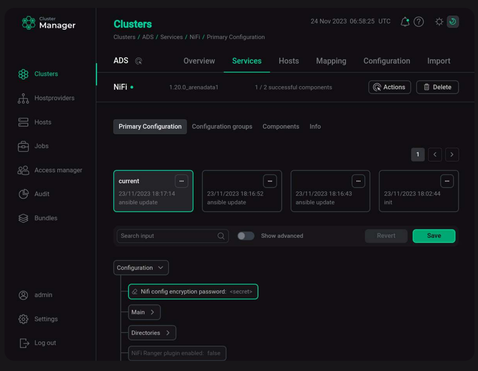
Шифрование для сервиса NiFi поддерживается при совершении следующих операций:

- Включение аутентификации и авторизации при помощи пользовательского интерфейса или плагина Ranger – при этом все новые конфиденциальные данные записываются в конфигурационные файлы в зашифрованном виде.
- Включение действия Expand – при этом существующий пароль используется для шифрования конфиденциальных данных в файлах конфигурации на новом хосте (или нескольких хостах).

Изменение пароля

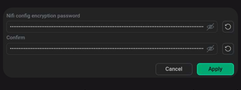
При необходимости изменение пароля выполняется в следующей последовательности:

1. На вкладке конфигурационных параметров сервиса NiFi в записи параметра ***Nifi config encryption password*** нажмите на ***<secret>***.



Пароль шифрования параметров конфигурации NiFi

2. В открывшемся окне введите новый пароль, подтвердите пароль и нажмите ***Apply***.



Изменение пароля шифрования параметров конфигурации NiFi

ВНИМАНИЕ

Пароль должен содержать не менее 12 символов и может содержать буквы, цифры и знаки препинания.

3. Сохраните конфигурацию, нажав ***Save***.
4. Перезагрузите сервис NiFi. Для этого примените действие Restart, нажав на иконку в столбце ***Actions***.

Сведения о шифровании конфиденциальных данных в файлах конфигурации NiFi

В таблице, приведенной ниже, указано, какие параметры в файлах конфигурации NiFi подвергаются шифрованию.

ВНИМАНИЕ

Таблица параметров предназначена только для информационных целей. Вносить изменения в файлы конфигурации не рекомендуется.

Компонент	Конфигурационный файл	Параметр	Значение по умолчанию	Зашифровано
NiFi Server	nifi.properties	nifi.sensitive.props.key	mysensetivekey	+
		nifi.security.keystorePasswd	—	+
		nifi.security.keyPasswd	—	+
		nifi.security.truststorePasswd	—	+
	nifi_cli.properties	keystorePasswd	—	—
		keyPasswd	—	—
		truststorePasswd	—	—
	login-identity-providers.xml	Manager Password	—	+
		TLS - Keystore Password	—	+
		TLS - Truststore Password	—	+
	authorizers.xml	Manager Password	—	+
		TLS - Keystore Password	—	+
		TLS - Truststore Password	—	+
NiFi Registry	nifi-registry.properties	nifi.registry.security.keystorePasswd	—	+
		nifi.registry.security.keyPasswd	—	+
		nifi.security.truststorePasswd	—	+
		nifi.registry.db.password	nifireg	+

Пример использования WP-REST API.

Оглавление

Оглавление
Создание нового проекта WP-REST API
Создание нового типа записи (post type) для нашего проекта WP-REST API
Создание нового поля (field) для нашего типа записи (post type)
Получение данных из нашего нового типа записи (post type) и нового поля (field) WP-REST API
Создание класса ProcMe
Получение данных из нашего типа записи ProcMe
Получение данных из нашего поля ProcMe
Получение нашего типа записи и поля ProcMe с использованием WP-REST API
Получение определенной записи (post) ProcMe
Получение определенного поля ProcMe
Получение определенного типа записи ProcMe
Получение определенного поля ProcMe
Получение определенного типа записи ProcMe с использованием WP-REST API
Получение определенного поля ProcMe с использованием WP-REST API
Запуск проекта/сайта с использованием WP-REST API
Получение определенного типа записи ProcMe с использованием WP-REST API
Получение определенного поля ProcMe с использованием WP-REST API
Создание класса Test
Получение данных из нашего типа записи Test
Получение данных из нашего поля Test
Получение нашего типа записи и поля Test с использованием WP-REST API
Получение определенной записи (post) Test
Получение определенного поля Test
Получение определенного типа записи Test
Получение определенного поля Test
Получение определенного типа записи Test с использованием WP-REST API
Получение определенного поля Test с использованием WP-REST API
Создание нового типа записи (post type) для нашего проекта WP-REST API
Получение данных из нашего нового типа записи (post type) WP-REST API
Создание нового поля (field) для нашего типа записи (post type)
Получение данных из нашего нового типа записи (post type) и нового поля (field) WP-REST API
Установка проекта/сайта
Изменение имени проекта/сайта
Авторизация пользователя с использованием WP-REST API

Создание нового проекта WP-REST API

Для создания нового проекта WP-REST API нам необходимо создать новый сайт. Для этого нам необходимо перейти в панель управления нашего хостинга (hosting) и создать новый сайт. Для этого нам необходимо перейти в раздел Сайты (Websites) и нажать кнопку Создать сайт (Create Website). Далее нам необходимо выбрать тип сайта. Для этого нам необходимо выбрать Пользовательский сайт (Custom Website). Далее нам необходимо выбрать версию PHP. Для этого нам необходимо выбрать PHP 8.1. Далее нам необходимо выбрать базу данных. Для этого нам необходимо выбрать MySQL 8.0. Далее нам необходимо нажать кнопку Создать (Create).

Создание нового типа записи (post type) для нашего проекта WP-REST API

Для создания нового типа записи (post type) для нашего проекта WP-REST API нам необходимо создать новый файл. Для этого нам необходимо перейти в папку нашего проекта/сайта и создать новый файл. Для этого нам необходимо перейти в папку wp-content/plugins и создать новый файл. Для этого нам необходимо перейти в папку wp-content/plugins/my-plugin и создать новый файл. Для этого нам необходимо перейти в папку wp-content/plugins/my-plugin/includes и создать новый файл. Для этого нам необходимо перейти в папку wp-content/plugins/my-plugin/includes/post-types и создать новый файл. Для этого нам необходимо перейти в папку wp-content/plugins/my-plugin/includes/post-types/my-post-type и создать новый файл. Для этого нам необходимо перейти в папку wp-content/plugins/my-plugin/includes/post-types/my-post-type/my-post-type.php

Построение потока данных

Содержание

Подготовка к созданию потока
Настройка постоянных параметров
Запись в поток данных из файла
Замена текста в готовом файле
Публикация содержимого потока в топик Kafka
Получение данных из топика Kafka
Разделение записей на несколько потоков
Запись содержимого потока в файл

Назад к содержанию

В данной статье мы начнем знакомство с [SSIS PowerPack](), который расширяет стандартные задачи

- Работа с потоками данных в файлах различных форматов
- Работа с топиками Kafka
- Работа с потоками данных в Azure

Для примеров будем использовать [SSIS Integration Toolkit for Microsoft Azure]() и [SSIS Integration Toolkit for Apache Kafka]() от [COZYROC]().

Подготовка к созданию потока

Для работы с потоками данных нам потребуется подготовить:

- Пакет SSIS, в котором будем создавать поток данных
- Подходящую задачу потока данных, например Data Flow Task
- Работа с потоками данных из файлов
  - Текстовый файл или файл формата [CSV]()
  - [Excel]() файл
  - [JSON]() файл
- Работа с топиками Kafka потребует [Kafka]()
- Подключение к Azure потребует [Azure]()

Обратите также внимание на [лицензирование]() данных компонентов.

Для работы с Kafka потребуется [Java]()

Также нам потребуется [.NET Framework]()

Для работы с Azure потребуется [Azure Feature Pack]()

Подготовка к созданию потока

Перейдем к созданию потока данных. Для этого откроем пакет SSIS.

- Пакет открывается в [Visual Studio]() или [SQL Server Data Tools]() в зависимости от версии SQL Server, с которой вы работаете.
- Создадим новый пакет SSIS
  - Например, [SSIS Tutorial: Create a Simple ETL Package]()
  - Или [SSIS Tutorial: Create a Package with the Data Flow Task]()

Добавляем новую задачу потока данных на поверхность конструктора. Для этого можно воспользоваться [панелью элементов]() или [окном SSIS Toolbox]() (если оно не отображается, его можно открыть через меню [View]() - [Other Windows]() - [SSIS Toolbox]()).

Перетаскиваем задачу потока данных на поверхность конструктора. Для этого можно воспользоваться [панелью элементов]() или [окном SSIS Toolbox]() (если оно не отображается, его можно открыть через меню [View]() - [Other Windows]() - [SSIS Toolbox]()).

Перенос задачи потока данных в Control Flow

Чтобы создать и управлять потоками данных, нам потребуется [SSIS PowerPack]().

Настройка постоянных параметров

Перейдем к общим настройкам компонентов потока данных.

- **Delay Validation** - разделение этапов на стадии проверки кода и выполнения. **FALSE** (default) - на этапе старта пакета проверка кода. **TRUE** - проверка кода на этапе выполнения.
- **Expose** - **TRUE** - параметры компонента доступны для настройки через [Parameters]() и [Expressions]().
- **DisableSSIS** - запуск или остановка задачи. Значение **TRUE** - задача не выполняется.

Для этого нам нужно перейти в [Control Flow]() и выбрать задачу потока данных.

Control Flow. Выбор задачи потока данных

Далее нам нужно перейти в [Properties]() и выбрать нужный параметр.

Properties. Выбор параметра

Чтобы открыть окно [Properties](), можно воспользоваться [горячими клавишами]() [Control] + [alt] + [F4], или [F4], или [View]() - [Properties Window]().

Запись в поток данных из файла

Запись в поток данных из файла можно произвести с помощью [компонентов]() Data Flow.

- [Flat File Source]() - [компонент]() Data Flow, который [читает]() данные из [текстового файла]() или [файла]() [CSV]()
- [Excel Source]() - [компонент]() Data Flow, который [читает]() данные из [файла]() [Excel]()
- [JSON Source]() - [компонент]() Data Flow, который [читает]() данные из [файла]() [JSON]()

Перейдем к [Control Flow]() и перенесем [Data Flow Task]() на [поверхность конструктора]().

Control Flow. Перенос Data Flow Task

Перейдем к [Data Flow]() и перенесем [Flat File Source]() на [поверхность конструктора]().

Data Flow. Перенос Flat File Source

Настроим [Flat File Source]():

- **Connection manager** - [менеджер]() [подключений]() - [создадим]() новое подключение
- **Columns** - [столбцы]() - [настроим]() [столбцы]()

Flat File Source. Настройка

Создадим новое подключение:

- **Connection manager type** - [тип]() [менеджера]() [подключений]() - **FLAT FILE**
- **Connection manager name** - [название]() [менеджера]() [подключений]() - **Source**
- **File name** - [имя]() [файла]() - **C:\Temp\Source.txt**

Flat File Connection Manager Editor. Создание подключения

Настроим [столбцы]():

- **Name** - [имя]() [столбца]() - **Value**
- **Data Type** - [тип]() [данных]() - **[DT_STR]**

Flat File Source. Настройка столбцов

Перенесем [Destination]() - [компонент]() Data Flow, который [записывает]() данные в [файл]() [SQL Server]() - [OLE DB Destination]().

Data Flow. Перенос OLE DB Destination

Настроим [OLE DB Destination]():

- **Connection manager** - [менеджер]() [подключений]() - [создадим]() новое подключение
- **Table or view** - [таблица]() или [представление]() - [создадим]() новую [таблицу]()

OLE DB Destination. Настройка

Создадим новое подключение:

- **Server name** - [имя]() [сервера]() - **(local)**
- **Database** - [база]() [данных]() - **Test**

OLE DB Connection Manager. Создание подключения

Создадим новую [таблицу]():

```
CREATE TABLE [dbo].[Destination](
	[Value] [varchar](max) NULL
)
```


Создание новой таблицы

Настроим [столбцы]():

- **Input Column** - [столбец]() [входного]() [потока]() - **Value**
- **Destination Column** - [столбец]() [таблицы]() [назначения]() - **Value**

OLE DB Destination. Настройка столбцов

Запустим [пакет]() на [выполнение]() - [нажмем]() [F5]

Запуск пакета на выполнение

Данные из [файла]() [перенесены]() в [таблицу]()

Данные перенесены в таблицу

Замена текста в готовом файле

Замена текста в готовом файле производится с помощью [компонентов]() Data Flow.

- [Script Component]() - [компонент]() Data Flow, который [может]() выполнять [код]() на [языке]() [C#]() или [VB]()

Создание нового скрипта Script Task

Настроим [Script Component]():

- **Script language** - [язык]() [скрипта]() - **Microsoft Visual C# 2012**
- **ReadOnlyVariables** - [переменные]() [Read-Only]() - **User::Value**

Script Transformation Editor. Настройка

Перейдем к [редактированию]() [скрипта]() - [нажмем]() [Edit Script...]

Редактирование скрипта

Добавим [код]() [скрипта]():

```
public override void Input0_ProcessInputRow(Input0Buffer Row)
{
    Row.Value = Row.Value.Replace("Source", "Destination");
}
```


Скрипт для замены текста

Запустим [пакет]() на [выполнение]() - [нажмем]() [F5]

Запуск пакета на выполнение

Текст [заменен]() в [файле]()

Текст заменен в файле

Публикация содержимого потока в топик Kafka

Публикация содержимого потока в топик Kafka производится с помощью [компонентов]() Data Flow.

- [Kafka Destination]() - [компонент]() Data Flow, который [записывает]() данные из [потока]() в [топик]() [Kafka]() [Kafka Destination]().

Создание нового скрипта PowerPivot

Настроим [Kafka Destination]():

- **Kafka Server** - [имя]() [сервера]() [Kafka]()
- **Kafka Version** - [версия]() [сервера]() [Kafka]()
- **Topic Name** - [имя]() [топика]() [Kafka]() [сервера]()
- **Input Columns** - [входные]() [столбцы]() - [выберем]() [столбцы]() [потока]() [данных]()

Kafka Destination. Настройка

Запустим [пакет]() на [выполнение]() - [нажмем]() [F5]

Запуск пакета на выполнение

Данные [опубликованы]() в [топике]() [Kafka]()

Данные опубликованы в топике Kafka

Получение данных из топика Kafka

Получение данных из топика Kafka производится с помощью [компонентов]() Data Flow.

- [Kafka Source]() - [компонент]() Data Flow, который [читает]() данные из [топика]() [Kafka]() [Kafka Source]().

Создание нового скрипта PowerPivot

Настроим [Kafka Source]():

- **Kafka Server** - [имя]() [сервера]() [Kafka]()
- **Kafka Version** - [версия]() [сервера]() [Kafka]()
- **Topic Name** - [имя]() [топика]() [Kafka]() [сервера]()
- **Group ID** - [группа]() [потребителей]() [топика]() [Kafka]()
- **Value Type** - [тип]() [данных]() [сообщений]() [топика]()

Kafka Source. Настройка

Запустим [пакет]() на [выполнение]() - [нажмем]() [F5]

Запуск пакета на выполнение

[Данные]() [получены]() из [топика]() [Kafka]()

Данные получены из топика Kafka

Разделение записей на несколько потоков

Разделение записей производится с помощью [компонентов]() Data Flow.

- [Conditional Split]() - [компонент]() Data Flow, который [разделяет]() [поток]() [данных]() [на]() [несколько]() [потоков]() [по]() [условию]() [Conditional Split]().

Создание нового скрипта PowerPivot

Настроим [Conditional Split]():

- **Output Name** - [имя]() [выходного]() [потока]() - **Value = 'Value1'**
- **Condition** - [условие]() - **[Value] == "Value1"**
- **Default output name** - [имя]() [потока]() [по]() [умолчанию]() - **Value2**

Conditional Split. Настройка

Запустим [пакет]() на [выполнение]() - [нажмем]() [F5]

Запуск пакета на выполнение

[Данные]() [разделены]() на [потоки]()

Данные разделены на потоки

Запись содержимого потока в файл

Для этого нужно воспользоваться [компонентами]() Data Flow.

- [Flat File Destination]() - [компонент]() Data Flow, который [записывает]() данные из [потока]() в [файл]() [CSV]()
- [Excel Destination]() - [компонент]() Data Flow, который [записывает]() данные из [потока]() в [файл]() [Excel]()
- [JSON Destination]() - [компонент]() Data Flow, который [записывает]() данные из [потока]() в [файл]() [JSON]()

Создадим [Excel Destination]():

- **Excel file path** - [путь]() [к]() [файлу]() - **C:\Temp\Destination.xlsx**
- **Excel version** - [версия]() [Excel]() - **Excel 2013**
- **Use Existing Excel File** - [использовать]() [существующий]() [файл]() - **FALSE**
- **Overwrite Excel file** - [перезаписать]() [файл]() - **TRUE**
- **Name of Excel worksheet** - [имя]() [листа]() [Excel]() - **Destination**

Excel Destination. Настройка

Создадим [Flat File Destination]():

- **Connection manager** - [менеджер]() [подключений]() - [создадим]() новое подключение
- **Mappings** - [сопоставления]() - [сопоставим]() [столбцы]()

Flat File Destination. Настройка

Создадим новое подключение:

- **Connection type** - [тип]() [менеджера]() [подключений]() - **FLAT FILE**
- **Connection manager name** - [название]() [менеджера]() [подключений]() - **Destination**
- **File name** - [имя]() [файла]() - **C:\Temp\Destination.txt**

Flat File Connection Manager. Создание подключения

Настроим [сопоставления]() [столбцов]():

Настройка сопоставления столбцов Flat File

Запустим [пакет]() на [выполнение]() - [нажмем]() [F5]

Запуск пакета на выполнение

[Данные]() [записаны]() в [файлы]()

Данные записаны в файлы

Используются [SSIS PowerPack]() и [SSIS Integration Toolkit for Apache Kafka]() от [COZYROC]().

Kafka

Работа с [Kafka]() [Java]() [Kafka Source]() [Kafka Destination]()

Azure

Работа с [Azure]() [Azure Feature Pack]()

Excel

Работа с [Excel]() [Excel Source]() [Excel Destination]()

JSON

Работа с [JSON]() [JSON Source]() [JSON Destination]()

Flat File

Работа с [текстовыми]() [файлами]() [Flat File Source]() [Flat File Destination]()

CSV

Работа с [CSV]() [Flat File Source]() [Flat File Destination]()

Script

Работа с [Script]() [Script Component]()

Data Flow

Работа с [Data Flow]() [Data Flow Task]()

Control Flow

Работа с [Control Flow]()

Обзор NiFi ADB Connector

Ольга Семме

Содержание

Назад к содержанию

Обзор

Архитектура NiFi ADB Connector

- PutGreenplumRecord
- StandartGpfdistService
- GreenplumDBCPConnectionPool

Поддерживаемые типы данных

Обзор

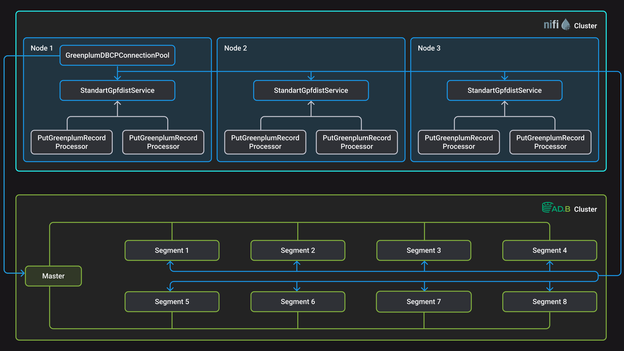
NiFi ADB Connector обеспечивает высокоскоростную параллельную запись данных из NiFi в Arenadata DB (ADB).

Начиная с ADS 3.9.0.1.b1 в пользовательском интерфейсе NiFi для создания NiFi ADB Connector доступны компоненты разработки Arenadata: процессор PutGreenplumRecord и сервис контроллера StandartGpfdistService.

Архитектура NiFi ADB Connector

Процессоры **PutGreenplumRecord** собирают данные и передают их в сервис контроллера **StandartGpfdistService**, который, используя встроенный протокол **gpfdist**, подключается к ADB через сервис пула подключений к базе данных **GreenplumDBCPConnectionPool**.

Greenplum Parallel File Server (gpfdist) — утилита Greenplum для чтения и записи данных из файлов, расположенных на удаленных серверах. Она установлена на всех хостах кластера ADB и обеспечивает параллельную загрузку данных, распределяя их между сегментами равномерно или согласно заданному ключу распределения данных.



Архитектура NiFi ADB Connector

PutGreenplumRecord

PutGreenplumRecord — процессор NiFi, обрабатывающий входные записи через приемник gpfdist, который создается StandartGpfdistService.

Ниже приведены параметры процессора.

Параметр	Обязательность	Описание	Значение по умолчанию
Gpfdist Service	true	Ссылка на запущенный сервис контроллера StandartGpfdistService	—
Record Reader	true	Ссылка на запуск одного из сервисов контроллера: CSVReader, AvroReader или другого, в зависимости от источника входных данных	—
Schema Name	false	Имя схемы, куда будут загружены данные	null
Table Name	true	Имя таблицы, в которую будут загружены данные	—
Table Columns	true	Столбцы таблицы, куда будут загружены данные	—

StandartGpfdistService

StandartGpfdistService — сервис контроллера для записи данных в сегменты ADB, использующий механизм читающей внешней таблицы с версией протокола gpfdist = 1.

Ниже приведены параметры сервиса контроллера.

Параметр	Обязательность	Описание	Значение по умолчанию
Listening Port	true	Порт для прослушивания входящих запросов gpfdist	—
Database Connection Pooling Service	true	Ссылка на настроенный сервис контроллера DBCPConnectionPool	—
Minimum Gpffist Server Threads	false	Минимальное количество потоков, используемых для запуска сервера gpfdist	1
Maximum Gpffist Server Threads	false	Максимальное количество потоков, используемых для запуска сервера gpfdist	4
The Maximum Gpffist Server Threads Idle Timeout	false	Максимальное время простоя потоков сервера gpfdist в миллисекундах	60000
Write buffer size in bytes	false	Размер буфера записи байтов для сериализованных записей	1Mb
Maximum Record Processor Threads	false	Максимальное количество потоков, используемых для обработки записей	8
Maximum Gpfdist Request Processor Threads	false	Максимальное количество потоков, используемых для обработки запроса gpfdist	8

GreenplumDBCPConnectionPool

GreenplumDBCPConnectionPool — сервис для подключения к ADB. Представляет собой сервис пула подключений к базе данных.

Поддерживаемые типы данных

Сопоставление типов данных ADB с типами полей записей NiFi приведено ниже.

Тип данных ADB	Тип данных поля записи NiFi	Комментарий
BIT	BOOLEAN	—
BOOLEAN	BOOLEAN	—
SMALLINT	SHORT, INT	Размер значения должен быть меньше 2 байт
INTEGER	INT	—
BIGINT	BIGINT, LONG	—
REAL	FLOAT	—
DOUBLE	DOUBLE	—
NUMERIC(p, s)	DECIMAL	—
CHAR(n)	STRING	—
VARCHAR(n)	STRING	—
ENUM	STRING	—
BYTEA	BYTE[], STRING	Для типа STRING значение должно быть записано в шестнадцатеричном формате; например: `\xd078`
DATE	DATE	—
TIME(n)	TIME	—
TIMESTAMP(n)	TIMESTAMP	—
TIMESTAMPTZ(n)	TIMESTAMP	—
MONEY	DECIMAL, STRING, DOUBLE, FLOAT	—
UUID	STRING	—
JSONB	STRING	—
HSTORE	STRING, MAP(STRING, STRING)	Для типа STRING значение должно быть в формате `key=value, key1=value1`
ARRAY	STRING, STRING[]	Для типа STRING значение должно быть в формате `{value, value1}`

Пример использования NiFi ADB Connector

Автор: ADS

Содержание

Назад к содержанию

Обзор
Предварительные требования
- ADS
- ADPG
- ADB
Подключение к PostgreSQL server (ADPG)
Подключение к Greenplum (ADB)
Запуск потока данных

Обзор

Для иллюстрации работы NiFi ADB Connector в статье показана реализация загрузки данных из таблиц базы данных PostgreSQL в таблицы ADB (на основе Greenplum).

Создание NiFi ADB Connector выполнялось в последовательности интерфейсов и доступно начиная с ADS 3.0.7.81.

Предварительные требования

Ниже описано окружение, используемое для создания NiFi ADB Connector.

ADS

- Кластер ADS развернут согласно руководству Online установки. Минимальная версия ADS — 3.0.7.81.
- Сервисы NiFi и Zookeeper установлены в кластере ADS.
- На хосте, где установлен NiFi, создана директория JDBC PostgreSQL, которая позволит программам подключаться к базам данных PostgreSQL, использующим стандартный код. Это нужно для работы Controller Service DBCPConnectionPool, для подключения к серверам PostgreSQL (ADPG) и Greenplum (ADB).
  Команда, приведенная ниже, позволяет загрузить последнюю версию JAR-файла (в данном примере используется версия 42.7.3) в нужную директорию:

```
[ mpp https://jdbc.postgresql.org/download/postgresql-42.7.3.jar
```

- При настройке сервиса DBCPConnectionPool также используется ссылка в формате `jdbc:postgresql://<hostname>:5432/<database>`, где:
  - `<hostname>` – хост, на котором установлен компонент ADPG (для подключения к мастеру PostgreSQL) или компонент ADB Master (для подключения к мастеру Greenplum);
  - `<database>` – наименование базы данных, в которой созданы используемые таблицы.

ADPG

- Кластер ADPG развернут согласно руководству Online установки.
- На установленном PostgreSQL server (кластер ADPG) выполнены подготовительные действия:
  - Пользователь с именем `my_user`, привилегиями `SUPERUSER` и паролем создан в ELT: `postgres`.
  - В ELT `public` создана таблица `My_Table` и в нее добавлены несколько строк с данными.
  - Файл pg_hba.conf настроен для обеспечения доступа пользователя с хоста, на котором установлен сервис NiFi кластера ADS. Для этого в поле `PG_HBA` на странице конфигурации параметров сервиса ADS добавлена запись об адресе хоста и пользователе:

```
host    postgres    my_user     10.92.41.206/32     trust
```

> **ПРИМЕЧАНИЕ**
>
> Для получения информации о работе с таблицей PostgreSQL обратитесь к статье Создание таблиц и загрузка через psql.
>
> Для получения информации о создании пользователя (роли) и назначении привилегий в базе данных PostgreSQL обратитесь к статье Роли и привилегии на базы данных.

ADB

- Кластер ADB развернут согласно руководству Online установки.
- Сервис ADB установлен в кластере ADB.
- Пользователь с именем `new_user`, привилегиями `SUPERUSER` и паролем создан в ELT: `adb`.
- В ELT `adb` создана таблица `My_Table` для записи данных.
- Файл pg_hba.conf настроен для обеспечения доступа пользователя с хоста, на котором установлен сервис NiFi кластера ADS. Для этого в поле `PG_HBA` на странице конфигурации параметров сервиса ADB добавлена запись об адресе хоста и пользователе:

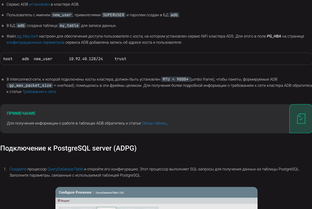
```
host    adb    new_user     10.92.40.128/24     trust
```

- В зависимости типа и которые подключены хосты кластера, должны быть установлен `RTS` и `MADH` (Apache Hadoop), чтобы пакеты, формируемые ADB `GG_DIR_INSTALLER` – метод(ы), помещались, и эти файлы целевом. Для получения более подробной информации о требованиях к сети кластера ADB обратитесь к статье Требования к сети.

> **ПРИМЕЧАНИЕ**
>
> Для получения информации о работе в таблице ADB обратитесь к статье Обзор таблиц.

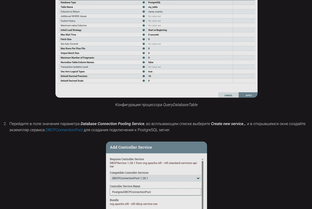
Подключение к PostgreSQL server (ADPG)

1. Создайте процессор QueryDatabaseTable и настройте его конфигурацию. Этот процессор выполняет SQL запросы для получения данных из таблиц PostgreSQL. Запросите параметры, связанные с используемой таблицей PostgreSQL.



Конфигурация процессора QueryDatabaseTable

2. Перейдите в поле значения параметра **Database Connection Pooling Service**, во всплывающем списке выберите **Create new service...**, и в открывшемся окне создайте экземпляр сервиса DBCPConnectionPool для создания подключения к PostgreSQL server.



Создание экземпляра сервиса DBCPConnectionPool

3. После создания созданного экземпляра нажмите кнопку и открывшемся окне **NiFi Flow Configuration** → **Controller Services** откройте конфигурацию сервиса и введите параметры, связанные с базой данных PostgreSQL.

Конфигурация сервиса PostgreSQLDBCPConnectionPool

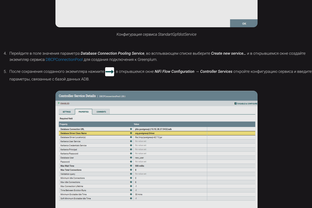
> **ПРИМЕЧАНИЕ**
>
> Создание экземпляра сервиса может быть выполнено до создания процессора. Это может быть использовано для подключения нескольких процессоров к одному сервису. Для создания сервиса выполните:
>
> 1. Правой кнопкой мыши кликните в пустом поле потока и выберите **Configure** в открывшемся контекстном меню.
> 2. В окне **NiFi Flow Configuration** перейдите на вкладку **Controller Services** и нажмите на **+**.
> 3. Выберите нужный сервис из списка и создайте экземпляр сервиса в открывшемся окне.

Подключение к Greenplum (ADB)

1. Для выполнения подключения к таблице Greenplum создайте процессор PutGreenplumRecord, откройте его конфигурацию и заполните параметры, связанные с таблицей Greenplum.

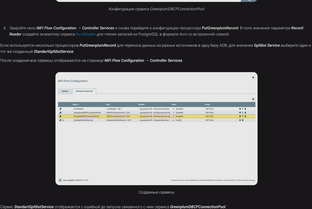
Конфигурация процессора PutGreenplumRecord

2. Перейдите в поле значения параметра **Upload Service**, во всплывающем списке выберите **Create new service...**, и в открывшемся окне создайте экземпляр сервиса StandardSqlWriterService.



Создание экземпляра сервиса StandardSqlWriterService

3. После создания созданного экземпляра нажмите кнопку и открывшемся окне **NiFi Flow Configuration** → **Controller Services** откройте конфигурацию сервиса и заполните необходимые параметры.



Конфигурация сервиса StandardSqlWriterService

4. Перейдите в поле значения параметра **Database Connection Pooling Service**, во всплывающем списке выберите **Create new service...**, и в открывшемся окне создайте экземпляр сервиса DBCPConnectionPool для создания подключения к Greenplum.

Создание экземпляра сервиса DBCPConnectionPool

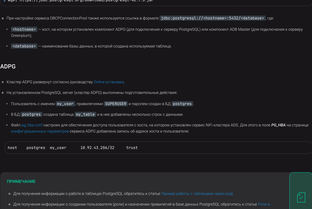
5. После создания созданного экземпляра нажмите кнопку и открывшемся окне **NiFi Flow Configuration** → **Controller Services** откройте конфигурацию сервиса и введите параметры, связанные с базой данных ADB.

Конфигурация сервиса GreenplumDBCPConnectionPool

6. Закройте окно **NiFi Flow Configuration** → **Controller Services** и снова перейдите к конфигурации процессора **PutGreenplumRecord**. В поле значения параметра **Record Reader** создайте экземпляр сервиса AvroReader для чтения записей из PostgreSQL в формате Avro со встроенной схемой.

Если использовать несколько процессоров **PutGreenplumRecord** для записи из разных источников в одну базу ADB, для каждого из них **Upload Service** выберите один и тот же созданный **StandardSqlWriterService**.

После создания все сервисы отображаются на странице **NiFi Flow Configuration** → **Controller Services**.



Созданные сервисы

Сервис **StandardSqlWriterService** отображается с ошибкой до запуска (связанного с ним сервиса **GreenplumDBCPConnectionPool**

Запуск потока данных

Создайте и настройте подключение между процессорами.

Созданные и соединенные процессоры

Процессоры отображаются с ошибками из-за того, что сервисы, связанные с ними, не запущены.

Для запуска потока:

1. Последовательно запустите сервисы на странице **NiFi Flow Configuration** → **Controller Services**, кликнув на иконку .
2. Запустите созданный поток данных.

Используя webpsql, в базе данных ADB, можно прочитать полученные данные.

Пример использования Schema Registry

Содержание

- Подготовка к работе
  - Установка Python3
  - Установка библиотек
- Подготовка кластера
- Создание и регистрация схемы AVRO
  - Создание схемы
  - Регистрация схемы
  - Получение данных
- Запись данных в топик Kafka с использованием схемы
- Чтение сообщений в соответствии со схемой
- Эволюция схемы

Текущая страница

Подготовка к работе

В данной статье я постараюсь описать пример использования Schema Registry. Схема - это описание структуры данных. Для чего это нужно? Представьте ситуацию, что у вас есть топик в Kafka, в который пишутся данные в формате JSON. Однажды вам захотелось добавить в эти данные новое поле. Вы можете столкнуться с тем, что ваши потребители не смогут обработать эти данные, потому что они ожидают старый формат. Чтобы избежать подобных ситуаций, и нужна Schema Registry.

Для примера я буду использовать Python. Для работы с Kafka из Python существует несколько библиотек. Я буду использовать confluent-kafka-python.

Также я буду использовать Avro. Avro - это формат сериализации данных, который позволяет описывать схему данных и обеспечивает их компактное хранение.

Установка Python3

Первое, что нам нужно сделать - установить Python 3. Для этого выполним следующие команды:

```
$ sudo yum install gcc openssl-devel bzip2-devel libffi-devel zlib-devel

$ cd /usr/src

$ sudo wget https://www.python.org/ftp/python/3.7.3/Python-3.7.3.tgz

$ sudo tar xzf Python-3.7.3.tgz

$ cd Python-3.7.3

$ sudo ./configure --enable-optimizations

$ sudo make altinstall
```

Проверим версию Python3 нужно при помощи команды:

```
$ python3 --version
```

Установка библиотек

Установим библиотеку confluent-kafka командой pip3:

```
$ sudo pip3 install confluent_kafka
```

Установим библиотеку requests также командой pip3:

```
$ sudo pip3 install requests
```

Установим остальные необходимые библиотеки:

```
$ sudo pip3 install fastavro
$ sudo pip3 install avro-python3
$ sudo pip3 install avro
$ sudo pip3 install pandas
```

Проверим, что все установилось:

```
$ pip3 freeze | grep -E "confluent-kafka|requests|fastavro|avro-python3|avro|pandas"
avro==1.9.0
avro-python3==1.9.0
confluent-kafka==1.0.0
fastavro==0.21.19
pandas==0.24.2
requests==2.21.0
```

Подготовка кластера

Для работы нам потребуется кластер Kafka, Schema Registry и Kafka Rest Proxy. Для развертывания кластера воспользуемся docker-compose. Создадим файл docker-compose.yml следующего содержания:

```
---
version: '2'

services:
  zookeeper:
    image: confluentinc/cp-zookeeper:5.2.1
    hostname: zookeeper
    container_name: zookeeper
    ports:
      - "2181:2181"
    environment:
      ZOOKEEPER_CLIENT_PORT: 2181
      ZOOKEEPER_TICK_TIME: 2000

  kafka:
    image: confluentinc/cp-kafka:5.2.1
    hostname: kafka
    container_name: kafka
    depends_on:
      - zookeeper
    ports:
      - "29092:29092"
      - "9092:9092"
    environment:
      KAFKA_BROKER_ID: 1
      KAFKA_ZOOKEEPER_CONNECT: "zookeeper:2181"
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:29092,PLAINTEXT_HOST://localhost:9092
      KAFKA_METRIC_REPORTERS: io.confluent.metrics.reporter.ConfluentMetricsReporter
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
      CONFLUENT_METRICS_REPORTER_BOOTSTRAP_SERVERS: kafka:29092
      CONFLUENT_METRICS_REPORTER_ZOOKEEPER_CONNECT: zookeeper:2181
      CONFLUENT_METRICS_REPORTER_TOPIC_REPLICAS: 1
      CONFLUENT_METRICS_ENABLE: 'true'
      CONFLUENT_SUPPORT_CUSTOMER_ID: 'anonymous'

  schema_registry:
    image: confluentinc/cp-schema-registry:5.2.1
    hostname: schema_registry
    container_name: schema_registry
    depends_on:
      - zookeeper
      - kafka
    ports:
      - "8081:8081"
    environment:
      SCHEMA_REGISTRY_HOST_NAME: schema_registry
      SCHEMA_REGISTRY_KAFKASTORE_CONNECTION_URL: 'zookeeper:2181'

  rest-proxy:
    image: confluentinc/cp-kafka-rest:5.2.1
    depends_on:
      - zookeeper
      - kafka
      - schema_registry
    ports:
      - 8082:8082
    hostname: rest-proxy
    container_name: rest-proxy
    environment:
      KAFKA_REST_HOST_NAME: rest-proxy
      KAFKA_REST_BOOTSTRAP_SERVERS: 'kafka:29092'
      KAFKA_REST_LISTENERS: "http://0.0.0.0:8082"
      KAFKA_REST_SCHEMA_REGISTRY_URL: 'http://schema_registry:8081'
      KAFKA_REST_CLIENT_SECURITY_PROTOCOL: 'PLAINTEXT'
```

Запустим кластер:

```
$ sudo docker-compose up -d
```

Проверим, что все запустилось:

```
$ sudo docker-compose ps
      Name                     Command               State                    Ports
-------------------------------------------------------------------------------------------------
kafka             /etc/confluent/docker/run        Up      0.0.0.0:29092->29092/tcp, 0.0.0.0:9092->9092/tcp
rest-proxy        /etc/confluent/docker/run        Up      0.0.0.0:8082->8082/tcp
schema_registry   /etc/confluent/docker/run        Up      0.0.0.0:8081->8081/tcp
zookeeper         /etc/confluent/docker/run        Up      0.0.0.0:2181->2181/tcp
```

Получение данных

В качестве данных для примера я буду использовать данные из сервиса Nominatim.

Для его работы нужно развернуть базу данных Postgis (здесь можно все посмотреть) и загрузить Полигоны России.

Затем я создал вьюху:

```
CREATE VIEW public.city_view AS
 SELECT place_id,
    name,
    address,
    country_code,
    ST_X(centroid) AS lon,
    ST_Y(centroid) AS lat
   FROM placex
  WHERE ((rank_address >= 13) AND (rank_address <= 25) AND (name IS NOT NULL) AND (indexed_status = 0) AND (linked_place_id IS NULL) AND ((osm_type = 'N'::character(1)) OR ((osm_type = 'R'::character(1)) AND (admin_level = ANY (ARRAY[5, 6, 7, 8])))) AND ((class = 'boundary'::text) OR (class = 'place'::text)) AND ((type = 'administrative'::text) OR (type = 'city'::text)) AND ((country_code = 'ru'::text) OR (country_code = 'by'::text) OR (country_code = 'kz'::text) OR (country_code = 'ua'::text)));
```

Которая позволяет получить данные в удобном для меня виде:

```
$ psql -h localhost -U postgres -d nominatim -c "SELECT * FROM city_view LIMIT 1;"
Password for user postgres:
 place_id |  name  |                                                
```

Действия в кластере

Ольга Семина

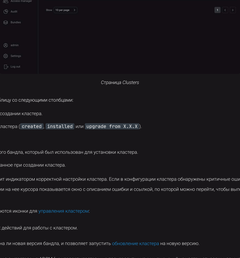
Содержание

Назад к содержанию

Управление кластером
Обновление кластера

В данной статье описаны основные действия, доступные в интерфейсе ADCM для управления кластером ADS.

Действия с кластером выполняются на странице **Clusters**.

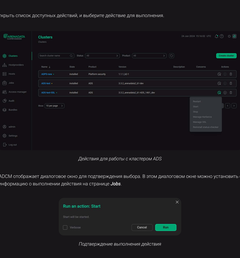


Страница Clusters

Страница **Clusters** содержит таблицу со следующими столбцами:

- **Name** — имя, указанное при создании кластера.
- **State** — текущее состояние кластера: `created`, `installed` или `upgrade from X.Y.X`.
- **Product** — имя продукта.
- **Version** — версия продуктового бандла, который был использован для установки кластера.
- **Description** — описание, указанное при создании кластера.
- **Concerns**. Иконка ① служит индикатором корректной настройки кластера. Если в конфигурации кластера обнаружены критичные ошибки, в столбце отображается иконка ⓘ — при наведении на нее курсора показывается окно с описанием ошибки и состоит, по которой можно перейти, чтобы выполнить требуемое конфигурирование.
- **Actions**. В столбце показываются иконки для управления кластером:
  - ⚙ — открывает список действий для работы с кластером.
  - ⟳ — указывает, доступна ли новая версия бандла, и позволяет запустить обновление кластера на новую версию.
  - 🗑 — удалит информацию о кластере из ADCM (не удаляет кластер и не производит никаких изменений на хостах, относящихся к кластеру).

Вы можете использовать иконки, описанные выше, для выполнения соответствующих действий. После того как действие запущено, ADCM отображает процесс его выполнения и результат на странице **Jobs**. В этой странице можно перейти на страницу отдельной задачи (шлюпку из имени задачи), чтобы увидеть внутренние этапы ее выполнения и проанализировать ошибки в случае их возникновения.



Страница задачи Install

ПРИМЕЧАНИЕ

При выполнении действий для сервисов в логах на странице **Jobs** могут отображаться сообщения типа `FAILED - RETRYING`, например:

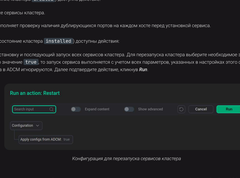
```
FAILED - RETRYING: Check NiFi Server service port (50 retries left).
FAILED - RETRYING: Check that the topic data is absent (40 retries left).
```

Это связано с тем, что запуск некоторых процессов может занять некоторое время. Если очередная попытка (retry) для текущего этапа действия (task) заканчивается успешно и действие продолжает выполняться и завершается в целом без ошибок, сообщения типа `FAILED - RETRYING` не являются признаком неуспешного результата.

В случае, если все попытки были выполнены и ни одна из них не привела к успеху, работа действия прекращается, и действие считается неуспешно проёденным.

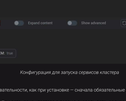
Управление кластером

Нажмите иконку ⚙, чтобы открыть список доступных действий, и выберите действие для выполнения.



Действия для работы с кластером ADS

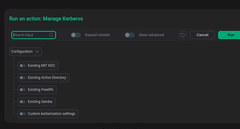
Когда вы выбираете действие, ADCM отображает диалоговое окно для подтверждения выбора. В этом диалоговом окне можно установить флажок **Verbose**, чтобы просмотреть дополнительную информацию о выполнении действия на странице **Jobs**.



Подтверждение выполнения действия

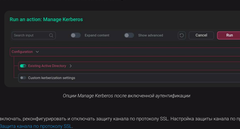
Набор доступных действий зависит от состояния кластера:

- Для нового кластера (состояние кластера `created`) доступно действие:
  - **Install** — устанавливает все сервисы кластера.
  - **Validate configuration** — выполняет проверку наличия дублирующихся портов на каждом хосте перед установкой сервиса.
- После установки кластера (состояние кластера `installed`) доступны действия:
  - **Restart** — осуществляет остановку и последующий запуск всех сервисов кластера. Для перезапуска кластера выберите необходимое значение опции **Apply configs from ADCM**. Если выбрано значение `true`, то запуск сервиса выполняется с учетом всех параметров, указанных в настройках этого сервиса в ADCM. В противном случае настройки сервиса в ADCM игнорируются. Далее подтвердите действие, кликнув **Run**.



Конфигурация для перезапуска сервисов кластера

- **Start** — осуществляет запуск сервисов кластера. Для запуска кластера выберите необходимое значение опции **Apply configs from ADCM** и подтвердите действие, кликнув **Run**.

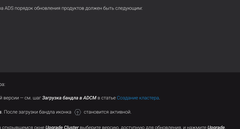


Конфигурация для запуска сервисов кластера

Запуск сервисов осуществляется в той же последовательности, как при установке — сначала обязательные сервисы, затем дополнительные.

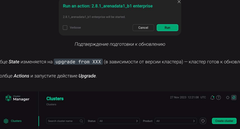
- **Stop** — осуществляет остановку сервисов кластера. При остановке кластера в первую очередь останавливаются дополнительные сервисы, затем обязательные.
- **Manage Kerberos** — позволяет включать, реконфигурировать и отключать аутентификацию по протоколу Kerberos.
  После выбора действия в появившемся окне будет предложено несколько опций для запуска действия:
  - Existing MIT KDC
  - Existing Active Directory
  - Existing FreeIPA
  - Existing Samba

  Каждую из опций действия **Manage Kerberos** можно совместить с опцией Custom kerberization settings.



Опции Manage Kerberos

После включенной аутентификации по одному из протоколов после выбора действия **Manage Kerberos** в открывшемся окне будет предложено редактирование опции, по которой уже выполнено включение, и опции **Custom kerberization settings**.



Опции Manage Kerberos после включенной аутентификации

- **Manage SSL** — позволяет включать, реконфигурировать и отключать защиту канала по протоколу SSL. Настройка защиты канала по протоколу SSL проводится в соответствии со статьей Защита канала по протоколу SSL.
- **Reinstall status-checker** — инициирует переустройку и перезапуск statuschecker для всех сервисов кластера. Используется в случае миграции кластера под управление нового сервера ADCM.
- **Validate configuration** — выполняет проверку наличия дублирующихся портов на каждом хосте.

- Для подготовленного к обновлению версии кластера (состояние кластера `upgrade from X.Y.X`) доступно действие:
  - **Upgrade** — выполняет обновление версии кластера.

Обновление кластера

ВАЖНО

- Перед обновлением убедитесь, что конфигурация на хостах соответствует конфигурациям ADCM. Если на хостах были ручные исправления, приведите их в соответствие согласно принятой самостоятельно.
- При обновлении кластера ADS порядок обновления продуктов должен быть следующим:

1. ADCM
2. ADPS
3. ADS

Для обновления версии кластера:

1. Загрузите бандл ADS новой версии — см. шаг **Загрузка бандла в ADCM** в статье Создание кластера.
2. Откройте страницу **Clusters**. После загрузки бандла иконка ⟳ становится активной.
3. Кликните на иконку ⟳, в открывшемся окне **Upgrade Cluster** выберите версию, доступную для обновления, и нажмите **Upgrade**

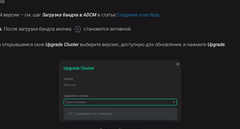
Запуск обновления

4. В открывшемся диалоговом окне кликните **Run**.

Подтверждение подготовки к обновлению

В результате статус в столбце **State** изменится на `upgrade from XXX` (в зависимости от версии кластера) — кластер готов к обновлению.

5. Кликните иконку ⚙ в столбце **Actions** и запустите действие **Upgrade**.



Выполнение обновления

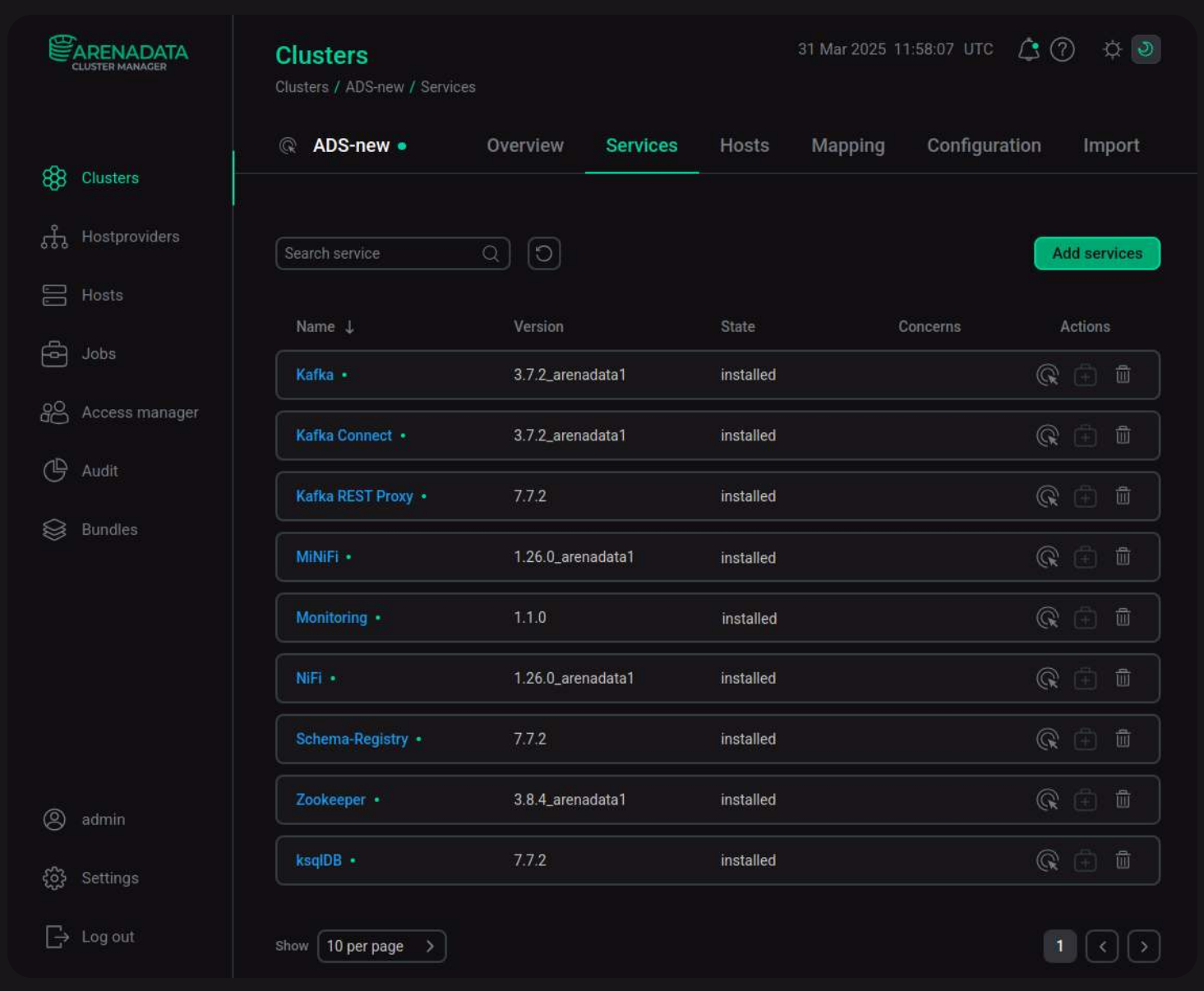
В открывшемся диалоговом окне нажмите **Run**, чтобы подтвердить выполнение обновления. Как только процесс обновления будет завершен, ADCM изменит состояние кластера на `installed`.

Действия в сервисе

Ольга Семме

Назад к содержанию

Пользовательский интерфейс ADCM позволяет выполнять действия для управления сервисами кластера ADS. Для перехода к действиям нажмите на имя кластера на странице ***Clusters*** и откройте вкладку ***Services***.



Сервисы кластера ADS в интерфейсе ADCM

Вкладка ***Services*** содержит таблицу со следующими столбцами:

- ***Name*** — имя сервиса.
- ***Version*** — версия сервиса.
- ***State*** — текущее состояние сервиса (например, `created` или `installed`).
- ***Concerns***. Иконка ⓘ служит индикатором корректной настройки компонентов сервиса. Если в конфигурации сервиса обнаружены критичные ошибки (например, обязательные компоненты сервиса не установлены на хостах кластера), в столбце отображается иконка ⓘ — при наведении на нее курсора показывается окно с описанием ошибки и ссылкой, по которой можно перейти, чтобы выполнить требуемое конфигурирование.
- ***Actions***. При нажатии на иконку открывается список действий для работы с сервисом.

  После того как действие запущено, ADCM отображает процесс и результат его выполнения на странице ***Jobs***. Также на странице ***Jobs*** можно проанализировать ошибки в случае их возникновения.

  Набор действий зависит от типа сервиса.

  Доступные действия для каждого сервиса описаны в статьях:

  - Действия в Kafka
  - Действия в Kafka Connect
  - Действия в Kafka REST Proxy
  - Действия в ksqlDB
  - Действия в NiFi
  - Действия в NiFi2
  - Действия в MiNiFi
  - Действия в Schema-Registry
  - Действия в ZooKeeper
  - Действия в Monitoring Clients
  - Действия в Monitoring

В столбце ***Actions*** также отдельно показываются иконки для следующих действий:

- — перевод сервиса в режим обслуживания (maintenance mode). В данный момент эта функциональность не реализована в данном кластере.
- — удаление сервиса из кластера.

При неудачном результате удаления сервиса на странице ***Jobs*** можно проанализировать причины ошибки. Некоторые сервисы взаимозависимы. Например, перед удалением некоторых сервисов необходимо удалить сервис Kafka.

Действия в Kafka

Ольга Семме

Содержание

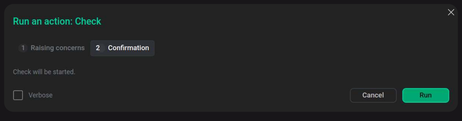
Назад к содержанию

- Действия с сервисом
  - Check
  - KRaft migration stage
  - Restart
  - Start
  - Stop
  - Add/Remove components
  - Manage Ranger plugin
  - Validate configuration
- Действия с компонентами
  - Restart
  - Start
  - Stop

В данной статье описаны основные действия, доступные для работы с сервисом в ADCM.

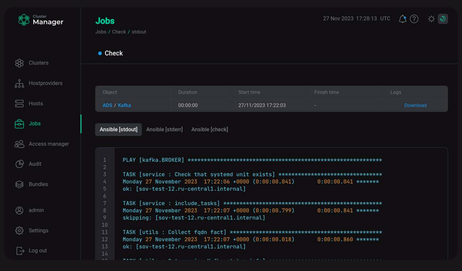
Чтобы перейти к действиям в сервисе, кликните по имени кластера на странице ***Clusters***, откройте вкладку ***Services***, нажмите на иконку в строке, соответствующей сервису Kafka, и выберите нужное действие.

Когда вы выбираете действие, ADCM отображает форму подтверждения. В этой форме на вкладке ***Confirmation*** можно установить флажок ***Verbose***, чтобы просмотреть дополнительную информацию о выполнении действия на странице ***Jobs***.



Форма подтверждения действия

Для просмотра результатов выполнения действия перейдите на страницу ***Jobs*** и кликните на название соответствующего действия. Далее становится доступна к просмотру страница с результатами выполнения действия, где можно проанализировать ошибки в случае их возникновения.



Результат действия

Чтобы перейти к действиям компонентов сервиса, со вкладки ***Services*** перейдите на страницу конфигурационных параметров сервиса, нажав на наименование сервиса.

Далее из вкладки ***Primary configuration*** перейдите на страницу конфигурирования компонентов сервиса.

Для выбора действия нажмите на иконку в строке, соответствующей компоненту.

Действия с сервисом

Check

В Kafka действие ***Check*** запускает создание тестовых топиков и проверяет их доступность на каждом из хостов с Kafka Broker.

KRaft migration stage

Действия ***KRaft migration stage 1*** и ***KRaft migration stage 2*** используются при миграции Kafka в режим KRaft.

Restart

Действие ***Restart*** осуществляет остановку и последующий запуск сервиса.

Для перезапуска сервиса:

1. Выберите необходимое значение опции ***Apply configs from ADCM***. Если выбрано значение `true`, то запуск сервиса выполняется с учетом всех параметров, указанных в настройках этого сервиса в ADCM. В противном случае настройки сервиса в ADCM игнорируются.
2. Выберите значение опции ***Rolling restart***. Если выбрано значение `true`, то при запуске действия будет применен последовательный перезапуск для запущенных приложений Java.

Start

Действие ***Start*** осуществляет запуск сервиса. Для запуска сервиса выберите необходимое значение опции ***Apply configs from ADCM***.

Stop

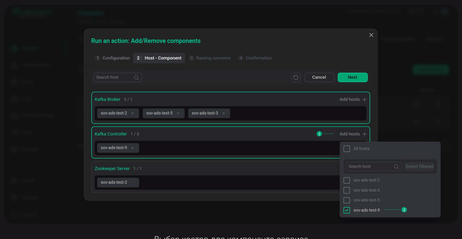
Действие ***Stop*** осуществляет остановку сервиса.

Add/Remove components

Действие ***Add/Remove components*** позволяет изменить количество компонентов, установленных для сервиса.

После выбора действия выполните:

1. Выберите значения параметров для установки.
2. Выберите хосты для установки экземпляров нового компонента или удалите хосты, сопоставленные ранее компоненту сервиса.



Выбор хостов для компонента сервиса

Manage Ranger plugin

Действие ***Manage Ranger plugin*** инициирует активацию плагина Ranger Kafka для включения авторизации сервиса Kafka.

Validate configuration

Действие ***Validate configuration*** выполняет проверку наличия дублирующихся портов на каждом хосте для сервиса.

Действия с компонентами

Restart

Действие ***Restart*** осуществляет остановку и последующий запуск компонента. После выбора действия укажите необходимое значение опции ***Apply configs from ADCM***. Если выбрано значение `true`, то действие выполняется с учетом всех параметров, указанных в настройках этого сервиса в ADCM. В противном случае настройки сервиса в ADCM игнорируются.

Start

Действие ***Start*** осуществляет запуск компонента. После выбора действия укажите необходимое значение опции ***Apply configs from ADCM***.

Stop

Действие ***Stop*** осуществляет остановку компонента.

Действия в Kafka Connect

Ольга Семме

Содержание

Назад к содержанию

Check

Restart

Start

Stop

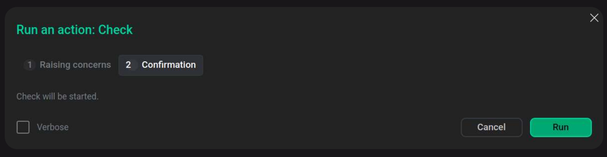
Add/Remove components

Validate configuration

В данной статье описаны основные действия, доступные для работы с сервисом в ADCM.

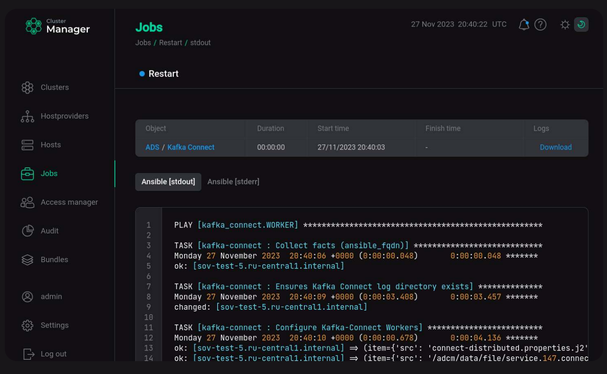
Чтобы перейти к действиям в сервисе, кликните по имени кластера на странице ***Clusters***, откройте вкладку ***Services***, нажмите на иконку в строке, соответствующей сервису Kafka Connect, и выберите нужное действие.

Когда вы выбираете действие, ADCM отображает форму подтверждения. В этой форме на вкладке ***Confirmation*** можно установить флажок ***Verbose***, чтобы просмотреть дополнительную информацию о выполнении действия на странице ***Jobs***.



Форма подтверждения действия

Для просмотра результатов выполнения действия перейдите на страницу ***Jobs*** и кликните на название соответствующего действия. Далее становится доступна к просмотру страница с результатами выполнения действия, где можно проанализировать ошибки в случае их возникновения.



Результат действия

Check

Действие ***Check*** осуществляет проверку работоспособности сервиса.

В Kafka Connect ***Check*** проверяет доступность сервиса на порте и статус сервиса.

Restart

Действие ***Restart*** осуществляет остановку и последующий запуск сервиса. Для перезапуска сервиса выберите необходимое значение опции ***Apply configs from ADCM***. Если выбрано значение `true`, то запуск сервиса выполняется с учетом всех параметров, указанных в настройках этого сервиса в ADCM. В противном случае настройки сервиса в ADCM игнорируются.

Start

Действие ***Start*** осуществляет запуск сервиса. Для запуска сервиса выберите необходимое значение опции ***Apply configs from ADCM***.

Stop

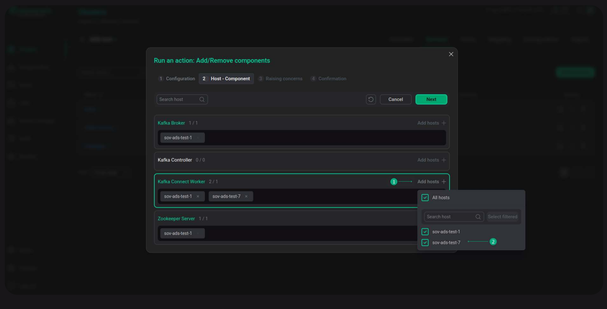
Действие ***Stop*** осуществляет остановку сервиса.

Add/Remove components

Действие ***Add/Remove components*** позволяет изменить количество компонентов, установленных для сервиса.

После выбора действия выполните:

1. Выберите значения параметров для установки.
2. Выберите хосты для установки экземпляров нового компонента или удалите хосты, сопоставленные ранее компоненту сервиса.



Выбор хостов для компонента сервиса

Validate configuration

Действие ***Validate configuration*** выполняет проверку наличия дублирующихся портов на каждом хосте для сервиса.

Действия в Kafka REST Proxy

Ольга Семме

Содержание

Назад к содержанию

Check

Restart

Start

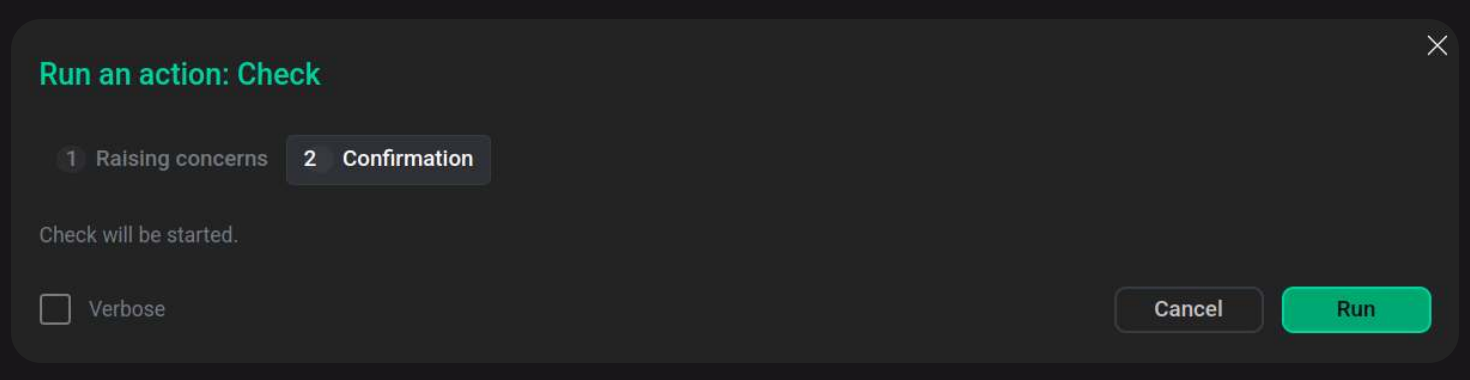
Stop

Validate configuration

В данной статье описаны основные действия, доступные для работы с сервисом в ADCM.

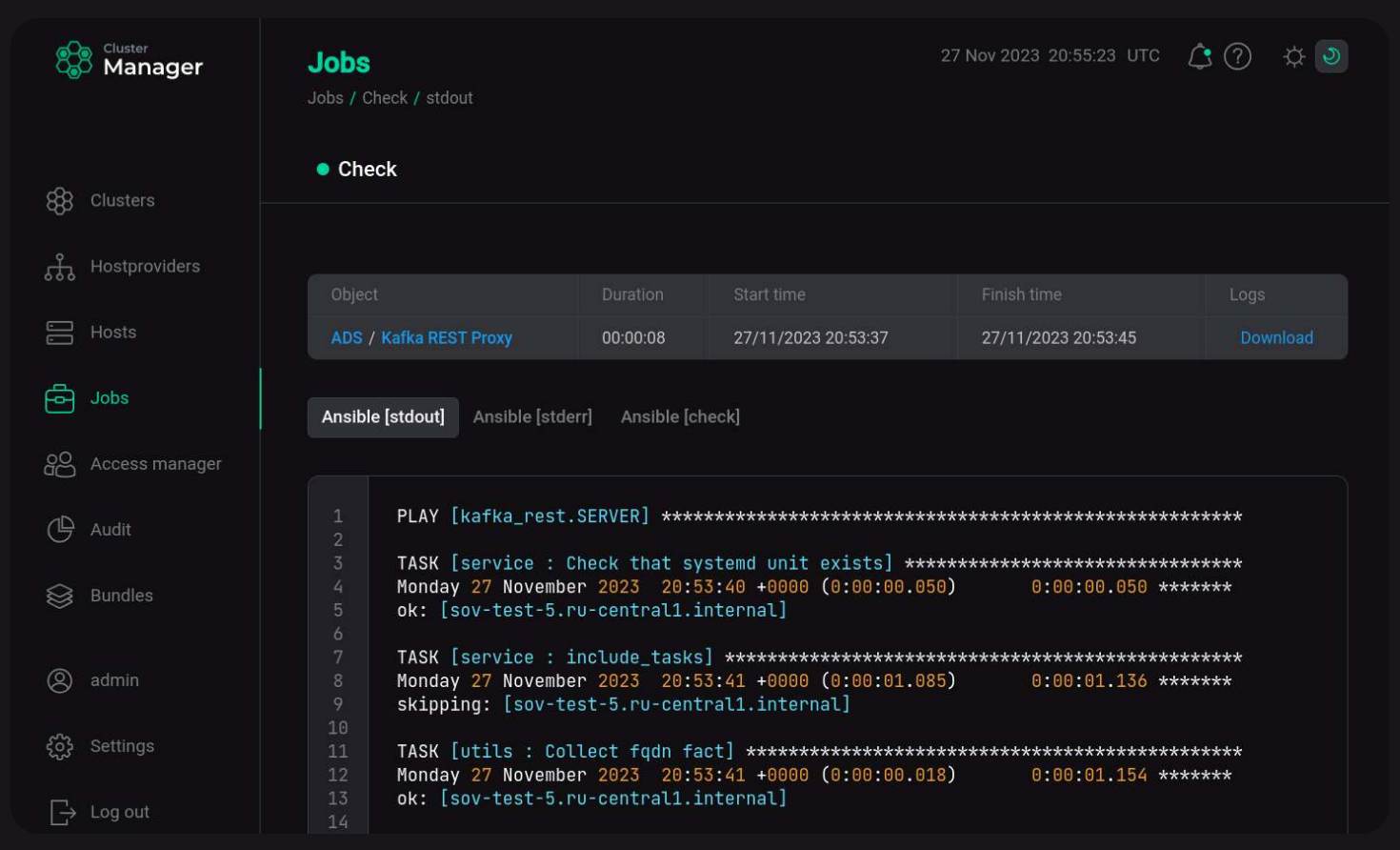
Чтобы перейти к действиям в сервисе, кликните по имени кластера на странице ***Clusters***, откройте вкладку ***Services***, нажмите на иконку в строке, соответствующей сервису Kafka REST Proxy, и выберите нужное действие.

Когда вы выбираете действие, ADCM отображает форму подтверждения. В этой форме на вкладке ***Confirmation*** можно установить флажок ***Verbose***, чтобы просмотреть дополнительную информацию о выполнении действия на странице ***Jobs***.



Форма подтверждения действия

Для просмотра результатов выполнения действия перейдите на страницу ***Jobs*** и кликните на название соответствующего действия. Далее становится доступна к просмотру страница с результатами выполнения действия, где можно проанализировать ошибки в случае их возникновения.



Результат действия

Check

Действие ***Check*** осуществляет проверку работоспособности сервиса.

В Kafka REST Proxy ***Check*** проверяет доступность сервиса на указанном порту и статус сервиса.

Restart

Действие ***Restart*** осуществляет остановку и последующий запуск сервиса. Для перезапуска сервиса выберите необходимое значение опции ***Apply configs from ADCM***. Если выбрано значение `true`, то запуск сервиса выполняется с учетом всех параметров, указанных в настройках этого сервиса в ADCM. В противном случае настройки сервиса в ADCM игнорируются.

Start

Действие ***Start*** осуществляет запуск сервиса. Для запуска сервиса выберите необходимое значение опции ***Apply configs from ADCM***.

Stop

Действие ***Stop*** осуществляет остановку сервиса.

Validate configuration

Действие ***Validate configuration*** выполняет проверку наличия дублирующихся портов на каждом хосте для сервиса.

Действия в ksqlDB

Ольга Семме

Содержание

Назад к содержанию

Check

Restart

Start

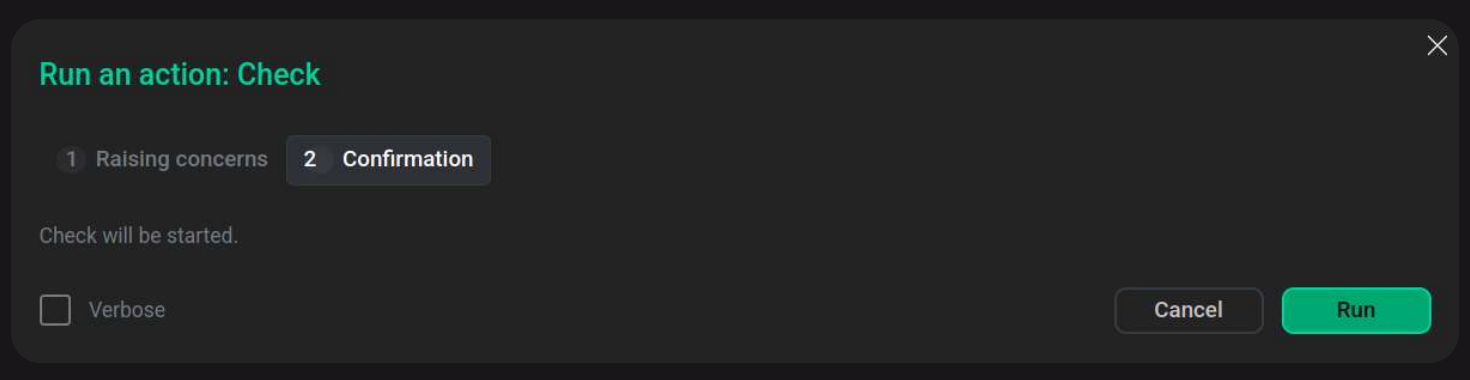
Stop

Validate configuration

В данной статье описаны основные действия, доступные для работы с сервисом в ADCM.

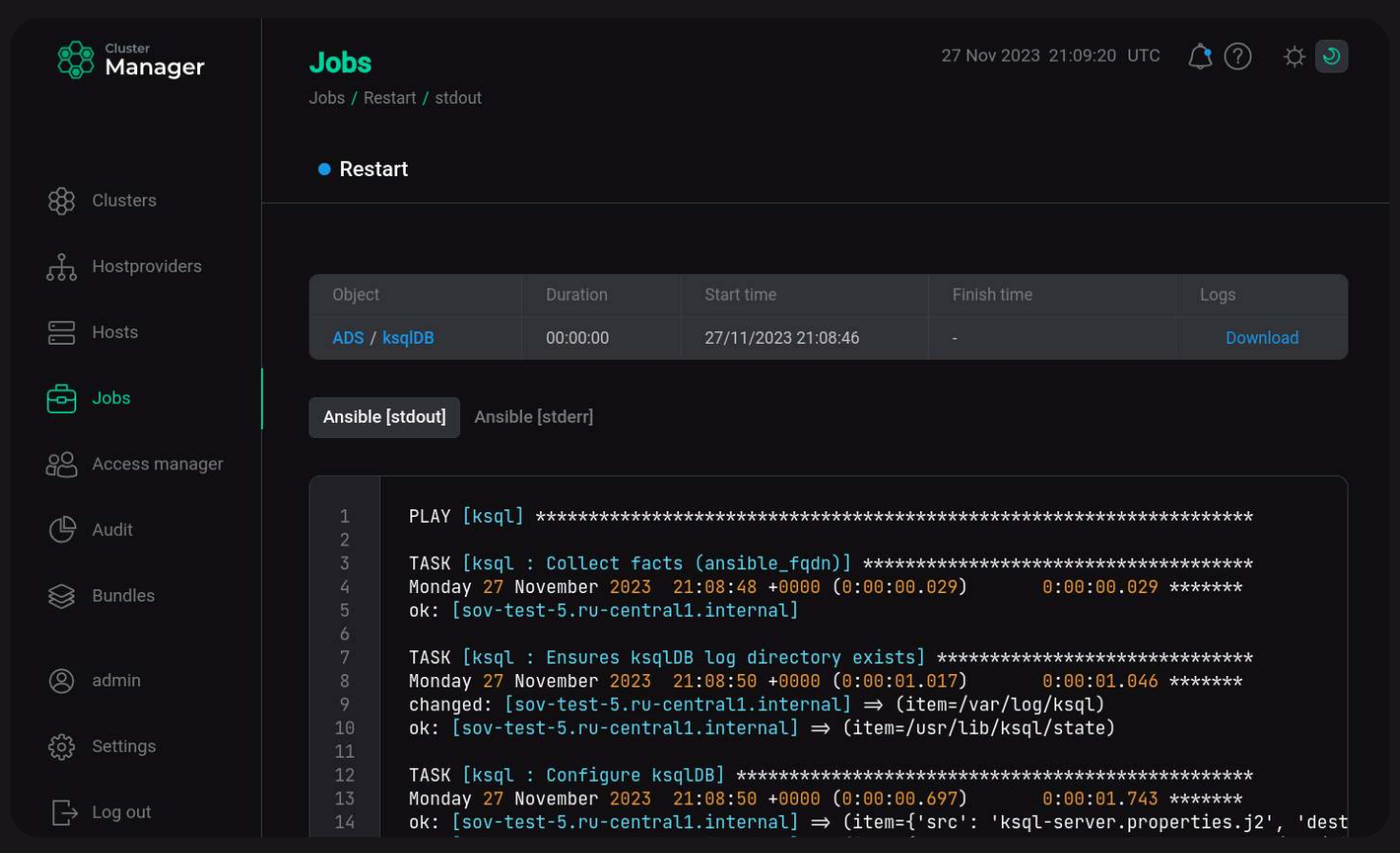
Чтобы перейти к действиям в сервисе, кликните по имени кластера на странице ***Clusters***, откройте вкладку ***Services***, нажмите на иконку в строке, соответствующей сервису ksqlDB, и выберите нужное действие.

Когда вы выбираете действие, ADCM отображает форму подтверждения. В этой форме на вкладке ***Confirmation*** можно установить флажок ***Verbose***, чтобы просмотреть дополнительную информацию о выполнении действия на странице ***Jobs***.



Форма подтверждения действия

Для просмотра результатов выполнения действия перейдите на страницу ***Jobs*** и кликните на название соответствующего действия. Далее становится доступна к просмотру страница с результатами выполнения действия, где можно проанализировать ошибки в случае их возникновения.



Результат действия

Check

Действие ***Check*** осуществляет проверку работоспособности сервиса.

В ksqlDB ***Check*** проверяет доступность сервиса на указанном порту и статус сервиса.

Restart

Действие ***Restart*** осуществляет остановку и последующий запуск сервиса. Для перезапуска сервиса выберите необходимое значение опции ***Apply configs from ADCM***. Если выбрано значение `true`, то запуск сервиса выполняется с учетом всех параметров, указанных в настройках этого сервиса в ADCM. В противном случае настройки сервиса в ADCM игнорируются.

Start

Действие ***Start*** осуществляет запуск сервиса. Для запуска сервиса выберите необходимое значение опции ***Apply configs from ADCM***.

Stop

Действие ***Stop*** осуществляет остановку сервиса.

Validate configuration

Действие ***Validate configuration*** выполняет проверку наличия дублирующихся портов на каждом хосте для сервиса.

Действия в NiFi

Ольга Семыкс

Содержание

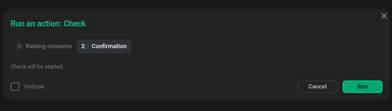
Назад к содержанию

- Действия с сервисом
  - Check
  - Kafka integration
  - Restart
  - Start
  - Stop
  - Add/Remove components
  - Manage Initial Admin
  - Manage Ranger plugin
  - Validate configuration
- Действия с компонентами
  - Restart
  - Start
  - Stop

В данной статье описаны основные действия, доступные для работы с сервисом в ADCM.

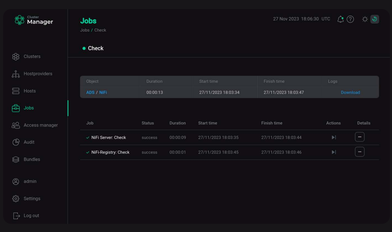
Чтобы перейти к действиям в сервисе, кликните по имени кластера на странице ***Clusters***, откройте вкладку ***Services***, нажмите на иконку в строке, соответствующей сервису NiFi, и выберите нужное действие.

Когда вы выбираете действие, ADCM отображает форму подтверждения. В этой форме на вкладке ***Confirmation*** можно установить флажок ***Verbose***, чтобы просмотреть дополнительную информацию о выполнении действия на странице ***Jobs***.



Форма подтверждения действия

Для просмотра результатов выполнения действия перейдите на страницу ***Jobs*** и кликните на название соответствующего действия. Далее становится доступна к просмотру страница с результатами выполнения действия, где можно проанализировать ошибки в случае их возникновения.



Результат действия

Чтобы перейти к действиям компонентов сервиса, со вкладки ***Services*** перейдите на страницу конфигурационных параметров сервиса, нажав на наименование сервиса.

Далее из вкладки ***Primary configuration*** перейдите на страницу конфигурирования компонентов сервиса.

Для выбора действия нажмите на иконку в строке, соответствующей компоненту.

ПРИМЕЧАНИЕ

При выполнении действий для сервиса ***NiFi*** в логах на странице ***Jobs*** могут отображаться сообщения типа `FAILED - RETRYING`, например:

```
FAILED - RETRYING: Check NiFi Server service port (50 retries left).
...
FAILED - RETRYING: Get Nifi-Registry info from NiFi Server (50 retries left).
```

Это связано с тем, что запуск серверов NiFi и NiFi Registry может занять некоторое время из-за особенностей их архитектуры. Если очередная попытка (retry) для текущего этапа действия (task) заканчивается успешно и действие продолжает выполняться и завершается в целом без ошибок, сообщения типа `FAILED - RETRYING` не являются признаком неуспешного результата.

В случае, если все попытки были выполнены и ни одна из них не привела к успеху, работа действия прекращается, и действие считается неуспешно пройденным.

Действия с сервисом

Check

В NiFi ***Check*** проверяет доступность сервиса на порте и статус сервиса для всех компонентов.

Kafka integration

Действие ***Kafka integration*** выполняет предварительную настройку интеграции между NiFi (NiFi2) и Kafka/Schema Registry. После запуска действия автоматически создается контекст параметров и экземпляр сервиса контроллера, которые отражают текущую топологию кластера и настройки безопасности.

Restart

Действие ***Restart*** осуществляет остановку и последующий запуск сервиса.

Для перезапуска сервиса выберите необходимое значение опции ***Apply configs from ADCM***. Если выбрано значение `true`, то запуск сервиса выполняется с учетом всех параметров, указанных в настройках этого сервиса в ADCM. В противном случае настройки сервиса в ADCM игнорируются.

Start

Действие ***Start*** осуществляет запуск сервиса. Для запуска сервиса выберите необходимое значение опции ***Apply configs from ADCM***.

Stop

Действие ***Stop*** осуществляет остановку сервиса.

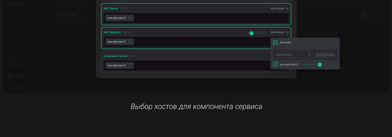
Add/Remove components

Действие ***Add/Remove components*** позволяет изменить количество компонентов, установленных для сервиса.

После выбора действия выполните:

1. Выберите значения параметров для установки.
   При этом заполните поля:
   - ***List of DNames to add*** — список идентификаторов пользователей и систем для добавления в NiFi Server/Registry.
   - ***List of DNames to remove*** — список идентификаторов пользователей и систем для удаления из NiFi Server/Registry.

   Для указания пользователя кликните символ ***+ 1*** и введите имя пользователя, например `CN=nifi_node_hostname, OU=Arenadata, O=Arenadata, L=Moscow, ST=Moscow, C=RU`.
2. Выберите хосты для установки экземпляров нового компонента или удалите хосты, сопоставленные ранее компоненту сервиса.



Выбор хостов для компонента сервиса

Manage Initial Admin

Действие ***Manage Initial Admin*** используется при необходимости изменения параметра ***NiFi Initial Admin*** после настройки аутентификации в сервисе NiFi.

Выполнение действия инициирует:

- Изменение параметра ***NiFi Initial Admin*** в разделе ***authorizers.xml*** конфигурационных параметров сервиса NiFi.
- Изменение данных для входа под администратором NiFi в пользовательский интерфейс NiFi Server и пользовательский интерфейс NiFi Registry.

Manage Ranger plugin

Действие ***Manage Ranger plugin*** инициирует активацию плагина Ranger NiFi для включения авторизации сервиса NiFi.

Validate configuration

Действие ***Validate configuration*** выполняет проверку наличия дублирующихся портов на каждом хосте для сервиса.

Действия с компонентами

Restart

Действие ***Restart*** осуществляет остановку и последующий запуск компонента.

Start

Действие ***Start*** осуществляет запуск компонента.

Stop

Действие ***Stop*** осуществляет остановку компонента

Действия в NiFi2

Ольга Семме

Содержание

Назад к содержанию

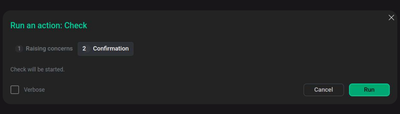
Действия с сервисом
- Check
- Kafka integration
- Restart
- Start
- Stop
- Add/Remove components
- Manage Initial Admin
- Validate configuration

Действия с компонентами
- Restart
- Start
- Stop

В данной статье описаны основные действия, доступные для работы с сервисом в ADCM.

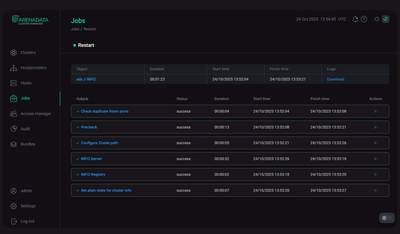
Чтобы перейти к действиям в сервисе, кликните по имени кластера на странице ***Clusters***, откройте вкладку ***Services***, нажмите на иконку в строке, соответствующей сервису NiFi2, и выберите нужное действие.

Когда вы выбираете действие, ADCM отображает форму подтверждения. В этой форме на вкладке ***Confirmation*** можно установить флажок ***Verbose***, чтобы просмотреть дополнительную информацию о выполнении действия на странице ***Jobs***.



Форма подтверждения действия

Для просмотра результатов выполнения действия перейдите на страницу ***Jobs*** и кликните на название соответствующего действия. Далее становится доступна к просмотру страница с результатами выполнения действия, где можно проанализировать ошибки в случае их возникновения.



Результат действия

Чтобы перейти к действиям компонентов сервиса, со вкладки ***Services*** перейдите на страницу конфигурационных параметров сервиса, нажав на наименование сервиса.

Далее из вкладки ***Primary configuration*** перейдите на страницу конфигурирования компонентов сервиса.

Для выбора действия нажмите на иконку в строке, соответствующей компоненту.

ПРИМЕЧАНИЕ

При выполнении действий для сервиса ***NiFi*** в логах на странице ***Jobs*** могут отображаться сообщения типа `FAILED - RETRYING`, например:

```
FAILED - RETRYING: Check NiFi Server service port (50 retries left).
...
FAILED - RETRYING: Get Nifi-Registry info from Nifi Server (50 retries left).
```

Это связано с тем, что запуск серверов NiFi и NiFi Registry может занять некоторое время из-за особенностей их архитектуры. Если очередная попытка (retry) для текущего этапа действия (task) заканчивается успешно и действие продолжает выполняться и завершается в целом без ошибок, сообщения типа `FAILED - RETRYING` не являются признаком неуспешного результата.

В случае, если все попытки были выполнены и ни одна из них не привела к успеху, работа действия прекращается, и действие считается неуспешно пройденным.

Действия с сервисом

Check

В NiFi2 ***Check*** проверяет доступность сервиса на порте и статус сервиса для всех компонентов.

Kafka integration

Действие ***Kafka integration*** выполняет предварительную настройку интеграции между NiFi (NiFi2) и Kafka/Schema Registry. После запуска действия автоматически создается контекст параметров и экземпляр сервиса контроллера, которые отражают текущую топологию кластера и настройки безопасности.

Restart

Действие ***Restart*** осуществляет остановку и последующий запуск сервиса.

Для перезапуска сервиса выберите необходимое значение опции ***Apply configs from ADCM***. Если выбрано значение `true`, то запуск сервиса выполняется с учетом всех параметров, указанных в настройках этого сервиса в ADCM. В противном случае настройки сервиса в ADCM игнорируются.

Start

Действие ***Start*** осуществляет запуск сервиса. Для запуска сервиса выберите необходимое значение опции ***Apply configs from ADCM***.

Stop

Действие ***Stop*** осуществляет остановку сервиса.

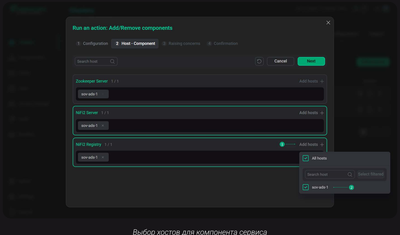
Add/Remove components

Действие ***Add/Remove components*** позволяет изменить количество компонентов, установленных для сервиса.

После выбора действия выполните:

1. Выберите значения параметров для установки.
   При этом заполните поля:
   - ***List of DNames to add*** — список идентификаторов пользователей и систем для добавления в NiFi Server/Registry.
   - ***List of DNames to remove*** — список идентификаторов пользователей и систем для удаления из NiFi Server/Registry.

   Для указания пользователя кликните символ **+ T** и введите имя пользователя, например `CN=nifi_node_hostname, OU=Arenadata, O=Arenadata, L=Moscow, ST=Moscow, C=RU`.
2. Выберите хосты для установки экземпляров нового компонента или удалите хосты, сопоставленные ранее компоненту сервиса.



Выбор хостов для компонента сервиса

Manage Initial Admin

Действие ***Manage Initial Admin*** используется при необходимости изменения параметра ***NiFi Initial Admin*** после настройки аутентификации в сервисе NiFi2.

Выполнение действия инициирует:

- Изменение параметра ***NiFi Initial Admin*** в разделе ***authorizers.xml*** конфигурационных параметров сервиса NiFi2.
- Изменение данных для входа под администратором NiFi2 в пользовательский интерфейс NiFi Server и пользовательский интерфейс NiFi Registry.

Validate configuration

Действие ***Validate configuration*** выполняет проверку наличия дублирующихся портов на каждом хосте для сервиса.

Действия с компонентами

Restart

Действие ***Restart*** осуществляет остановку и последующий запуск компонента.

Start

Действие ***Start*** осуществляет запуск компонента.

Stop

Действие ***Stop*** осуществляет остановку компонента

Действия в MiNiFi

Ольга Семме

Содержание

Назад к содержанию

Check

Restart

Start

Stop

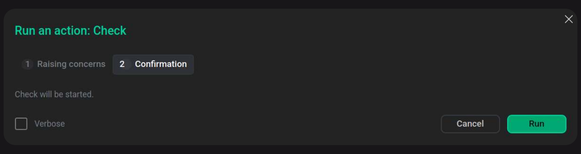
Add/Remove components

Validate configuration

В данной статье описаны основные действия, доступные для работы с сервисом в ADCM.

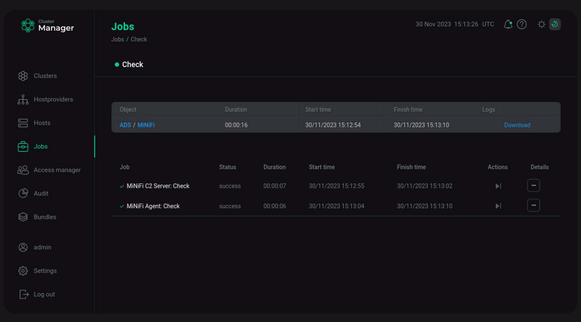
Чтобы перейти к действиям в сервисе, кликните по имени кластера на странице ***Clusters***, откройте вкладку ***Services***, нажмите на иконку в строке, соответствующей сервису MiNiFi, и выберите нужное действие.

Когда вы выбираете действие, ADCM отображает форму подтверждения. В этой форме на вкладке ***Confirmation*** можно установить флажок ***Verbose***, чтобы просмотреть дополнительную информацию о выполнении действия на странице ***Jobs***.



Форма подтверждения действия

Для просмотра результатов выполнения действия перейдите на страницу ***Jobs*** и кликните на название соответствующего действия. Далее становится доступна к просмотру страница с результатами выполнения действия, где можно проанализировать ошибки в случае их возникновения.



Результат действия

Check

Действие ***Check*** осуществляет проверку работоспособности сервиса.

В MiNiFi ***Check*** проверяет доступность сервиса на порте и статус сервиса для каждого компонента.

Restart

Действие ***Restart*** осуществляет остановку и последующий запуск сервиса. Для перезапуска сервиса выберите необходимое значение опции ***Apply configs from ADCM***. Если выбрано значение `true`, то запуск сервиса выполняется с учетом всех параметров, указанных в настройках этого сервиса в ADCM. В противном случае настройки сервиса в ADCM игнорируются.

Start

Действие ***Start*** осуществляет запуск сервиса. Для запуска сервиса выберите необходимое значение опции ***Apply configs from ADCM***.

Stop

Действие ***Stop*** осуществляет остановку сервиса.

Add/Remove components

Действие ***Add/Remove components*** позволяет изменить количество компонентов, установленных для сервиса.

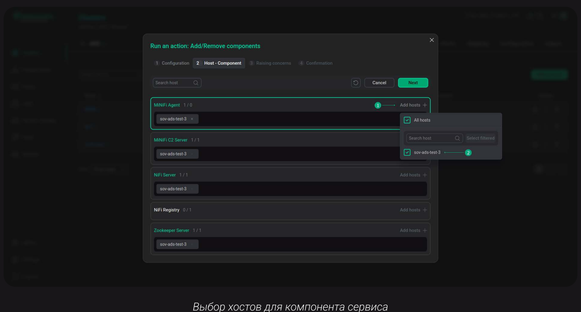
После выбора действия выполните:

1. Выберите значения параметров для установки.
   При этом заполните поля:

   - ***List of DNames to add*** — список идентификаторов пользователей MiNiFi и систем для добавления в NiFi Server/Registry.
   - ***List of DNames to remove*** — список идентификаторов пользователей MiNiFi и систем для удаления из NiFi Server/Registry.

   Для указания пользователя кликните символ ***+ 1*** и введите имя пользователя, например `CN=nifi_node_hostname, OU=Arenadata, O=Arenadata, L=Moscow, ST=Moscow, C=RU`.

2. Выберите хосты для установки экземпляров нового компонента или удалите хосты, сопоставленные ранее компоненту сервиса.



Выбор хостов для компонента сервиса

Validate configuration

Действие ***Validate configuration*** выполняет проверку наличия дублирующихся портов на каждом хосте для сервиса.

Действия в Schema-Registry

Ольга Семме

Содержание

Назад к содержанию

Check

Restart

Start

Stop

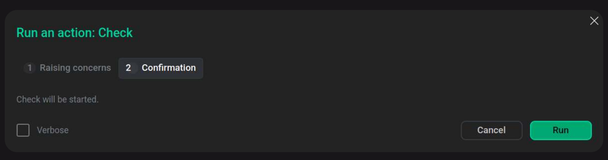
Add/Remove components

Validate configuration

В данной статье описаны основные действия, доступные для работы с сервисом в ADCM.

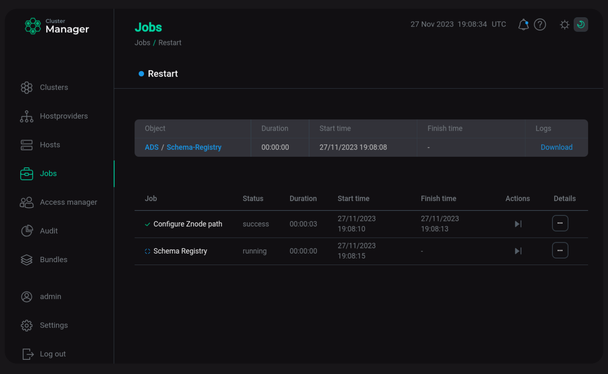
Чтобы перейти к действиям в сервисе, кликните по имени кластера на странице ***Clusters***, откройте вкладку ***Services***, нажмите на иконку в строке, соответствующей сервису Schema-Registry, и выберите нужное действие.

Когда вы выбираете действие, ADCM отображает форму подтверждения. В этой форме на вкладке ***Confirmation*** можно установить флажок ***Verbose***, чтобы просмотреть дополнительную информацию о выполнении действия на странице ***Jobs***.



Форма подтверждения действия

Для просмотра результатов выполнения действия перейдите на страницу ***Jobs*** и кликните на название соответствующего действия. Далее становится доступна к просмотру страница с результатами выполнения действия, где можно проанализировать ошибки в случае их возникновения.



Результат действия

Check

Действие ***Check*** осуществляет проверку работоспособности сервиса.

В Schema-Registry ***Check*** запускает создание тестового топика, записывает в него сообщения и проверяет доступность сообщений на каждом из хостов с Kafka Broker.

Restart

Действие ***Restart*** осуществляет остановку и последующий запуск сервиса. Для перезапуска сервиса выберите необходимое значение опции ***Apply configs from ADCM***. Если выбрано значение `true`, то запуск сервиса выполняется с учетом всех параметров, указанных в настройках этого сервиса в ADCM. В противном случае настройки сервиса в ADCM игнорируются.

Start

Действие ***Start*** осуществляет запуск сервиса. Для запуска сервиса выберите необходимое значение опции ***Apply configs from ADCM***.

Stop

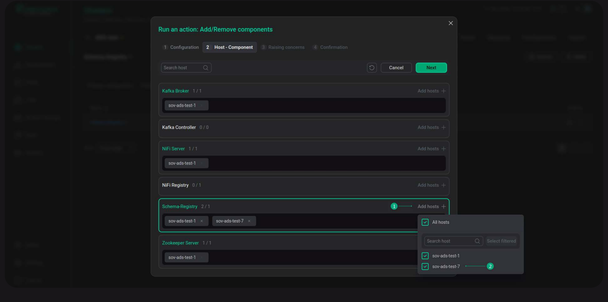
Действие ***Stop*** осуществляет остановку сервиса.

Add/Remove components

Действие ***Add/Remove components*** позволяет изменить количество компонентов, установленных для сервиса.

После выбора действия выполните:

1. Выберите значения параметров для установки.
2. Выберите хосты для установки экземпляров нового компонента или удалите хосты, сопоставленные ранее компоненту сервиса.



Выбор хостов для компонента сервиса

Validate configuration

Действие ***Validate configuration*** выполняет проверку наличия дублирующихся портов на каждом хосте для сервиса.

Действия в ZooKeeper

Ольга Семме

Содержание

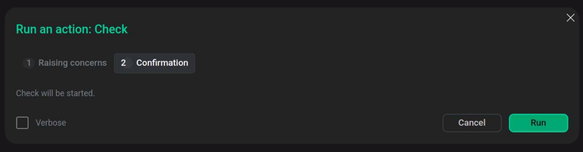
Назад к содержанию

Check
Restart
Start
Stop
Add/Remove components
Validate configuration

В данной статье описаны основные действия, доступные для работы с сервисом в ADCM.

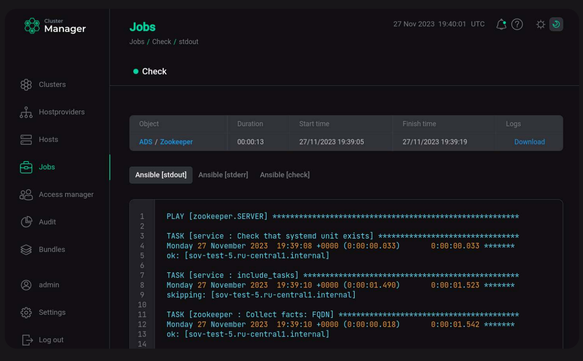
Чтобы перейти к действиям в сервисе, кликните по имени кластера на странице ***Clusters***, откройте вкладку ***Services***, нажмите на иконку в строке, соответствующей сервису ZooKeeper, и выберите нужное действие.

Когда вы выбираете действие, ADCM отображает форму подтверждения. В этой форме на вкладке ***Confirmation*** можно установить флажок ***Verbose***, чтобы просмотреть дополнительную информацию о выполнении действия на странице ***Jobs***.



Форма подтверждения действия

Для просмотра результатов выполнения действия перейдите на страницу ***Jobs*** и кликните на название соответствующего действия. Далее становится доступна к просмотру страница с результатами выполнения действия, где можно проанализировать ошибки в случае их возникновения.



Результат действия

Check

Действие ***Check*** осуществляет проверку работоспособности сервиса.

В ZooKeeper ***Check*** выполняет подключение к кворуму ZooKeeper, создание в нем тестовой znode и проверку доступности созданной znode каждому из хостов кворума.

Restart

Действие ***Restart*** осуществляет остановку и последующий запуск сервиса. Для перезапуска сервиса выберите необходимое значение опции ***Apply configs from ADCM***. Если выбрано значение `true`, то запуск сервиса выполняется с учетом всех параметров, указанных в настройках этого сервиса в ADCM. В противном случае настройки сервиса в ADCM игнорируются.

Start

Действие ***Start*** осуществляет запуск сервиса. Для запуска сервиса выберите необходимое значение опции ***Apply configs from ADCM***.

Stop

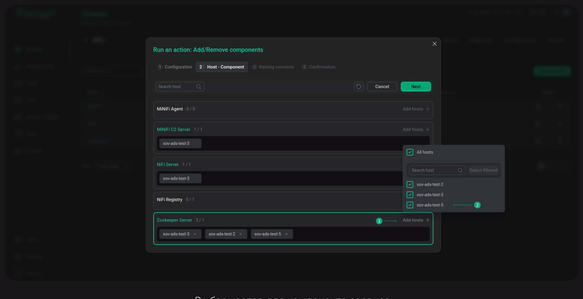
Действие ***Stop*** осуществляет остановку сервиса.

Add/Remove components

Действие ***Add/Remove components*** позволяет изменить количество компонентов, установленных для сервиса.

После выбора действия выполните:

1. Выберите значения параметров для установки.
2. Выберите хосты для установки экземпляров нового компонента или удалите хосты, сопоставленные ранее компоненту сервиса.



Выбор хостов для компонента сервиса

ПРИМЕЧАНИЕ

Количество экземпляров ZooKeeper должно быть нечетным.

Validate configuration

Действие ***Validate configuration*** выполняет проверку наличия дублирующихся портов на каждом хосте для сервиса.

Действия в Monitoring Clients

Ольга Семме

Содержание

Назад к содержанию

Check

Restart

Start

Stop

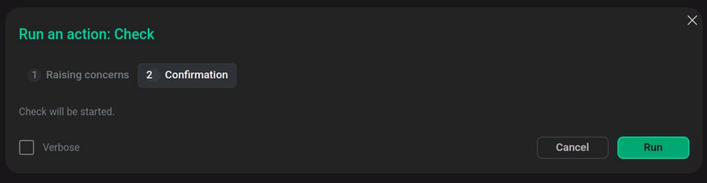
Add/Remove components

Reinstall

В данной статье описаны основные действия, доступные для работы с сервисом в ADCM.

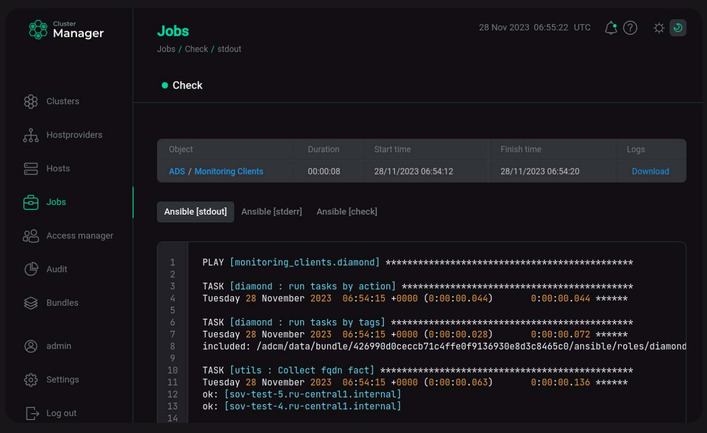
Чтобы перейти к действиям в сервисе, кликните по имени кластера на странице ***Clusters***, откройте вкладку ***Services***, нажмите на иконку в строке, соответствующей сервису Monitoring Clients, и выберите нужное действие.

Когда вы выбираете действие, ADCM отображает форму подтверждения. В этой форме на вкладке ***Confirmation*** можно установить флажок ***Verbose***, чтобы просмотреть дополнительную информацию о выполнении действия на странице ***Jobs***.



Форма подтверждения действия

Для просмотра результатов выполнения действия перейдите на страницу ***Jobs*** и кликните на название соответствующего действия. Далее становится доступна к просмотру страница с результатами выполнения действия, где можно проанализировать ошибки в случае их возникновения.



Результат действия

Check

Действие ***Check*** осуществляет проверку работоспособности сервиса.

В Monitoring Clients ***Check*** проверяет статусы всех компонентов сервиса.

Restart

Действие ***Restart*** осуществляет остановку и последующий запуск сервиса. Для перезапуска сервиса выберите необходимое значение опции ***Apply configs from ADCM***. Если выбрано значение `true`, то запуск сервиса выполняется с учетом всех параметров, указанных в настройках этого сервиса в ADCM. В противном случае настройки сервиса в ADCM игнорируются.

Start

Действие ***Start*** осуществляет запуск сервиса. Для запуска сервиса выберите необходимое значение опции ***Apply configs from ADCM***.

Stop

Действие ***Stop*** осуществляет остановку сервиса.

Add/Remove components

Действие ***Add/Remove components*** позволяет изменить количество компонентов, установленных для сервиса.

После выбора действия выполните:

1. Выберите значения параметров для установки.
2. Выберите хосты для установки экземпляров нового компонента или удалите хосты, сопоставленные ранее компоненту сервиса.

Reinstall

Действие ***Reinstall*** осуществляет переустановку Monitoring Clients. После запуска действия установите параметры и кликните ***Run***.

Действия в Monitoring

Ольга Семме

Содержание

Назад к содержанию

Add/Remove components

Check

Reconfigure and restart

Reinstall

Start

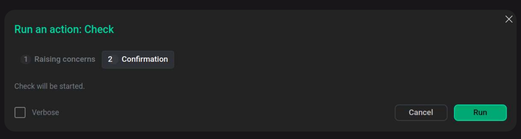
Stop

Validate configuration

В данной статье описаны основные действия, доступные для работы с сервисом в ADCM.

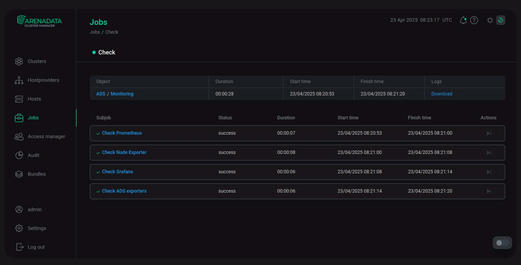
Чтобы перейти к действиям в сервисе, кликните по имени кластера на странице ***Clusters***, откройте вкладку ***Services***, нажмите на иконку в строке, соответствующей сервису Monitoring, и выберите нужное действие.

Когда вы выбираете действие, ADCM отображает форму подтверждения. В этой форме на вкладке ***Confirmation*** можно установить флажок ***Verbose***, чтобы просмотреть дополнительную информацию о выполнении действия на странице ***Jobs***.



Форма подтверждения действия

Для просмотра результатов выполнения действия перейдите на страницу ***Jobs*** и кликните на название соответствующего действия. Далее становится доступна к просмотру страница с результатами выполнения действия, где можно проанализировать ошибки в случае их возникновения.

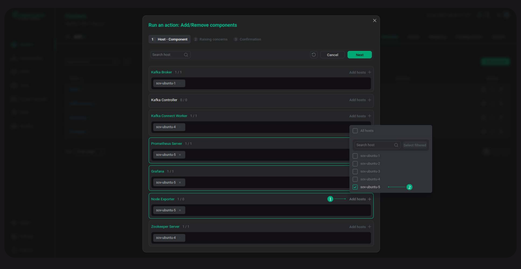


Результат действия

Add/Remove components

Действие ***Add/Remove components*** позволяет изменить количество компонентов, установленных для сервиса.

После выбора действия выберите хосты для установки экземпляров нового компонента или удалите хосты, сопоставленные ранее компоненту сервиса.



Выбор хостов для компонента сервиса

> **ПРИМЕЧАНИЕ**
>
> Компоненты ***Prometheus Server*** и ***Grafana*** могут быть установлены только по одному экземпляру на кластер.

Check

Действие ***Check*** осуществляет проверку работоспособности сервиса.

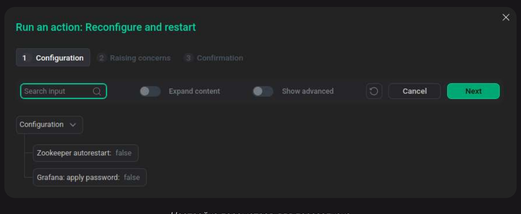
В Monitoring действие ***Check*** проверяет статусы всех компонентов сервиса.

Reconfigure and restart

Действие ***Reconfigure and restart*** позволяет перенастроить сервис, если были изменены конфигурационные параметры, а также останавливать и снова запускать сервис.

После выбора действия при необходимости выберите значения параметров:

- ***Zookeeper autorestart*** — установите в `true`, если следует перенастроить и перезапустить сервис ZooKeeper.
- ***Grafana: apply password*** — установите в `true`, если необходимо переустановить пароль для компонента Grafana.



Настройка параметров для перезапуска

Reinstall

Действие ***Reinstall*** осуществляет переустановку Monitoring.

Start

Действие ***Start*** осуществляет запуск сервиса.

Stop

Действие ***Stop*** осуществляет остановку сервиса.

Validate configuration

Действие ***Validate configuration*** выполняет проверку наличия дублирующихся портов на каждом хосте для сервиса.

Basic-аутентификация

Ольга Семме

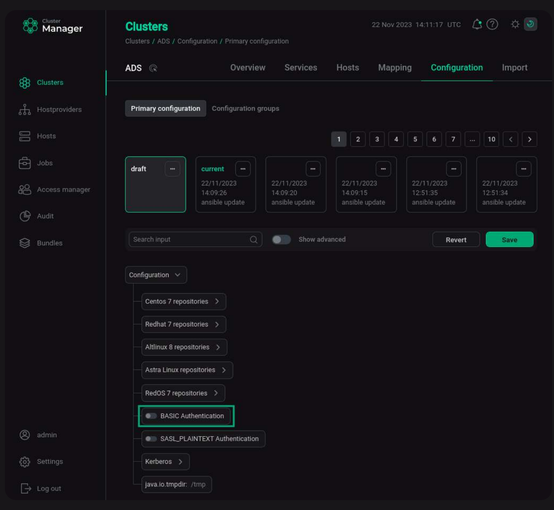
Назад к содержанию

В данной статье описано включение базовой (basic) аутентификации для сервисов:

- Schema Registry
- Kafka REST Proxy
- ksqlDB

Для включения базовой аутентификации с помощью ADCM выполните шаги:

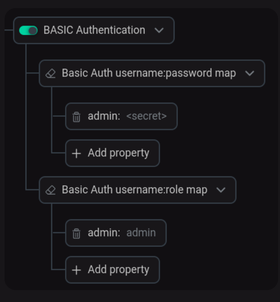
1. На вкладке конфигурирования кластера в дереве конфигурационных настроек включите переключатель ***BASIC Authentication***.



Включение BASIC Authentication в кластере ADS

В открывшихся узлах указаны значения по умолчанию для пользователя `admin`:

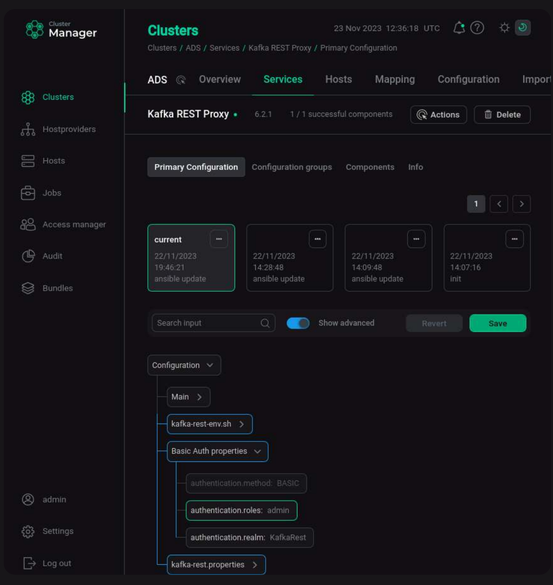
- в узле ***Basic Auth username:password map:*** — зашифрованный пароль для пользователя (по умолчанию – `admin`);
- в узле ***Basic Auth username:role map:*** — роль, назначенная пользователю (`admin`).



Конфигурация BASIC Authentication

Роли пользователя

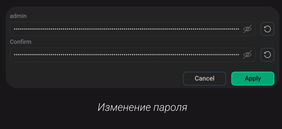
Роли, которые могут быть назначены пользователям, определяются на вкладке конфигурирования сервиса в дереве конфигурационных настроек при помощи параметра ***authentication.roles*** в узле ***Basic Auth properties***.



Параметр authentication.roles

Может быть указано несколько ролей, разделенных запятыми. Чтобы авторизоваться для работы в сервисе, аутентифицированный пользователь должен принадлежать хотя бы к одной из этих ролей.

2. При необходимости изменения пароля пользователя `admin` нажмите на ***<secret>*** рядом с именем, введите новый пароль, подтвердите пароль и нажмите ***Apply***.



Изменение пароля

3. Сохраните конфигурацию, нажав ***Save***.
4. Перезагрузите все сервисы, для которых была настроена аутентификация. Для этого примените действие Restart, нажав на иконку в столбце ***Actions***. После успешного применения действия ***Basic Auth*** подключение к сервисам производится после аутентификации указанного пользователя.

MIT Kerberos

Илья Космач

Содержание

Настройка KDC

Настройка файлов конфигурации

Настройка пользовательского jass.conf

Запуск MIT Kerberos KDC на кластере ADS в интерфейсе ADCM

Назад к содержанию

Kerberos - сетевой протокол аутентификации, который обеспечивает механизм взаимной аутентификации клиента и сервера перед установлением связи между ними. **KDC** - Kerberos Key Distribution Center, служба, которая выдает клиентам и серверам ключи сессии.

Основные термины, которые нам понадобятся для настройки Kerberos:

- **Принципал (principal)** - уникальное имя для клиента, которому разрешена аутентификация.
- **Хост (host)** - сервер, на котором работает служба для аутентификации и защиты, на которое клиенты хотят войти.
- **Область (realm)** - логическая область, которая управляется одним KDC с некоторой информацией о принципалах.
- **Ключ для принципалов хостов (Key Distribution Center, KDC)** - общий секретный ключ клиента.

Настройка KDC

Установка krb5-server

- KDC может быть установлен на любом хосте, который имеет доступ к его устанавливаемым KDC (рекомендуется подключение по TCP/IP).
- KDC для разных областей может быть развернут на одном хосте.

Для установки KDC на выбранном хосте выполните команду:

```
sudo yum install -y krb5-server
```

В результате выведется сообщение:

```
Complete!
```

Настройка файлов конфигурации

Основные настройки Kerberos хранятся в файлах конфигурации /etc/krb5.conf и /var/kerberos/krb5kdc/kdc.conf. Давайте откроем и отредактируем эти файлы.

Откроем /etc/krb5.conf на редактирование:

```
sudo vim /var/kerberos/krb5kdc/kdc.conf
```

Содержимое файла конфигурации:

```
[libdefaults]
 dns_lookup_realm = false
 ticket_lifetime = 24h
 renew_lifetime = 7d
 forwardable = true
 rdns = false
 default_realm = ADS.MYKERBEROS.LOCAL

[realms]
 ADS.MYKERBEROS.LOCAL = {
  kdc = vlan1.mykerberos.mydomain.ru
  admin_server = vlan1.mykerberos.mydomain.ru
 }

[domain_realm]
 .mykerberos.mydomain.ru = ADS.MYKERBEROS.LOCAL
 mykerberos.mydomain.ru = ADS.MYKERBEROS.LOCAL
```

[libdefaults]

В данном разделе указываются значения по умолчанию для Kerberos, которые будут использоваться, если другие [realms] не заданы явным для этого хоста.

Тут	Описание тут
kdc_ports	UDP-порт для подключения к KDC
kdc_tcp_ports	TCP-порт для подключения к KDC
default_realm	Область, указанная пользователями для проведения аутентификации

[realms]

Здесь задаются и указаны - какие области Kerberos. Например, для каждой области могут быть свои KDC и административный сервер.

Тут	Описание тут
ad_file	Расположение файла с ключами для службы, которая будет использовать аутентификацию. Ключи, хранящиеся в этом файле, используются для проверки запросов аутентификации, которые служба получает от клиентов.
admin_keytab	Расположение файла keytab, который служба администрирования Kerberos будет использовать для обслуживания запросов изменения пароля
dict_file	Расположение файла словаря, содержащего слова, которые нельзя использовать в качестве паролей. Если файл существует, сервер паролей Kerberos отклоняет любой пароль, который совпадает со словом в файле (без учета регистра). Формат каждого файла словаря - один текст на строку.
supported_enctypes	Определяет комбинации типов ключей шифрования и типов контрольных сумм, которые Kerberos будет использовать с этой областью. Каждый тип ключа шифрования, поддерживаемый Kerberos, должен иметь записи для этой области.

[logging]

Здесь задаются и указаны - как обстоят дела с ведением журнала Kerberos.

Тут	Описание тут
default	Путь к файлу, где сведения журнала про отсутствие других явных указаний
kdc	Путь к файлу, где находятся журналы для KDC
admin_server	Путь к файлу, где находятся журналы для административного сервера

[dbdefaults]

Здесь задаются значения по параметрам, используемым по умолчанию для баз данных.

Тут	Описание тут
default_realm	Для каждой базы данных есть область, к которой она относится. Если это значение не задано, будет использоваться область по умолчанию, указанная в разделе [libdefaults].
kdb_library	Для поддержки нескольких баз данных в KDC можно использовать несколько библиотек. Каждая библиотека должна быть указана в этом параметре, и должна существовать интерфейсная точка входа с именем kdb_<value>_initialize(). Например, для библиотеки db2 должна существовать интерфейсная точка входа kdb_db2_initialize().
keyfile_stashfile	Устанавливает файлы хранения по умолчанию для основных ключей. Если это значение не задано, будет использоваться файл по умолчанию, указанный в разделе [libdefaults].

[realms]

Каждый подраздел в разделе - имя области Kerberos. Значение тега определяет параметры для этой основной области.

Тут	Описание тут
ADS.MYKERBEROS.LOCAL	Область, используемая пользователями для проведения аутентификации
admin_server	Идентифицирует хост, на котором работает сервер администрирования. Как правило, это главный сервер KDC, но это не обязательно. Сервер администрирования должен быть запущен на указанном здесь хосте.
kdc	Имя или адрес хоста, на котором работает KDC для этой области. Может быть несколько необязательных записей тега, поддерживает IP-имена и номера портов.

Настройка файлов конфигурации

Основные настройки Kerberos хранятся в файлах конфигурации /etc/krb5.conf и /var/kerberos/krb5kdc/kdc.conf. Давайте откроем и отредактируем эти файлы.

Откроем /etc/krb5.conf на редактирование:

```
sudo vim /etc/krb5.conf
```

Содержимое файла конфигурации:

```
[libdefaults]
 default = FILE:/var/log/krb5libs.log
 kdc = FILE:/var/log/krb5kdc.log
 admin_server = FILE:/var/log/kadmind.log

[libdefaults]
 dns_lookup_realm = false
 ticket_lifetime = 24h
 renew_lifetime = 7d
 forwardable = true
 rdns = false
 default_realm = ADS.MYKERBEROS.LOCAL

[realms]
 ADS.MYKERBEROS.LOCAL = {
  kdc = vlan1.mykerberos.mydomain.ru
  admin_server = vlan1.mykerberos.mydomain.ru
 }

[domain_realm]
 .mykerberos.mydomain.ru = ADS.MYKERBEROS.LOCAL
 mykerberos.mydomain.ru = ADS.MYKERBEROS.LOCAL
```

[libdefaults]

Задает указанные ниже опции Kerberos:

Тут	Описание тут
default	Путь к файлу, где сведения журнала про отсутствие других явных указаний
kdc	Путь к файлу, где находятся журналы для KDC
admin_server	Путь к файлу, где находятся журналы для административного сервера

[dbdefaults]

Задает значения по параметрам, используемым по умолчанию для баз данных.

Тут	Описание тут
default_realm	Идентифицирует хост, на котором работает сервер администрирования. Как правило, это главный сервер KDC, но это не обязательно. Сервер администрирования должен быть запущен на указанном здесь хосте.
kdb_library	Для поддержки нескольких баз данных в KDC можно использовать несколько библиотек. Каждая библиотека должна быть указана в этом параметре, и должна существовать интерфейсная точка входа с именем kdb_<value>_initialize(). Например, для библиотеки db2 должна существовать интерфейсная точка входа kdb_db2_initialize().
keyfile_stashfile	Устанавливает файлы хранения по умолчанию для основных ключей. Если это значение не задано, будет использоваться файл по умолчанию, указанный в разделе [libdefaults].
kdc_ports	Устанавливает номер порта, который KDC будет использовать для UDP-связи. Значение по умолчанию - 88.
kdc_tcp_ports	Устанавливает номер порта, который KDC будет использовать для TCP-связи. Значение по умолчанию - 88.

[realms]

Каждый подраздел в разделе - имя области Kerberos. Значение тега определяет параметры для этой основной области.

Тут	Описание тут
ADS.MYKERBEROS.LOCAL	Область, используемая пользователями для проведения аутентификации
admin_server	Идентифицирует хост, на котором работает сервер администрирования. Как правило, это главный сервер KDC, но это не обязательно. Сервер администрирования должен быть запущен на указанном здесь хосте.
kdc	Имя или адрес хоста, на котором работает KDC для этой области. Может быть несколько необязательных записей тега, поддерживает IP-имена и номера портов.

Настройка пользовательского jass.conf

Создайте файл данных пользователя Kerberos для этого пользователя и сохраните его в /var/kerberos/krb5kdc/. Например, /var/kerberos/krb5kdc/client.keytab.

Введите команду:

```
sudo /usr/sbin/kadmin.local -q "addprinc -randkey client/admin@ADS.MYKERBEROS.LOCAL"
```

В результате выведется сообщение:

```
Authenticating as principal root/admin@ADS.MYKERBEROS.LOCAL with password.
WARNING: no policy specified for client/admin@ADS.MYKERBEROS.LOCAL; defaulting to no policy
Principal "client/admin@ADS.MYKERBEROS.LOCAL" created.
```

Создайте keytab-файл для этого принципала:

```
sudo /usr/sbin/kadmin.local -q "ktadd -k /var/kerberos/krb5kdc/client.keytab client/admin@ADS.MYKERBEROS.LOCAL"
```

В результате выведется сообщение:

```
Authenticating as principal root/admin@ADS.MYKERBEROS.LOCAL with password.
Entry for principal client/admin@ADS.MYKERBEROS.LOCAL with kvno 2, encryption type aes256-cts-hmac-sha1-96 added to keytab WRFILE:/var/kerberos/krb5kdc/client.keytab.
Entry for principal client/admin@ADS.MYKERBEROS.LOCAL with kvno 2, encryption type aes128-cts-hmac-sha1-96 added to keytab WRFILE:/var/kerberos/krb5kdc/client.keytab.
```

Перенесите /var/kerberos/krb5kdc/client.keytab на хост, где установлен Kafka Connect.

Перенесите /var/kerberos/krb5kdc/krb5.conf на хост, где установлен Kafka Connect.

Установите права доступа:

```
sudo chmod 777 /var/kerberos/krb5kdc/client.keytab
sudo chmod 777 /etc/krb5.conf
```

Создайте файл jass.conf и сохраните его в /etc/kafka-connect-ads/conf/. Например, /etc/kafka-connect-ads/conf/jass.conf.

Введите команду:

```
sudo nano /etc/kafka-connect-ads/conf/jass.conf
```

В результате выведется содержимое файла:

```
KafkaClient {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  storeKey=true
  keyTab="/var/kerberos/krb5kdc/client.keytab"
  principal="client/admin@ADS.MYKERBEROS.LOCAL";
};
```

Создайте файл jass.conf и сохраните его в /etc/kafka-connect-ads/conf/. Например, /etc/kafka-connect-ads/conf/jass.conf.

Введите команду:

```
sudo nano /etc/kafka-connect-ads/conf/jass.conf
```

В результате выведется содержимое файла:

```
KafkaClient {
  com.sun.security.auth.module.Krb5LoginModule required
  useKeyTab=true
  storeKey=true
  keyTab="/var/kerberos/krb5kdc/client.keytab"
  principal="client/admin@ADS.MYKERBEROS.LOCAL";
};
```

Создайте файл client_config.conf и сохраните его в /etc/kafka-connect-ads/conf/. Например, /etc/kafka-connect-ads/conf/client_config.conf.

Введите команду:

```
sudo nano /etc/kafka-connect-ads/conf/client_config.conf
```

В результате выведется содержимое файла:

```
security.protocol=SASL_PLAINTEXT
sasl.kerberos.service.name=kafka
sasl.jaas.config=com.sun.security.auth.module.Krb5LoginModule required \
  useKeyTab=true \
  storeKey=true \
  keyTab="/var/kerberos/krb5kdc/client.keytab" \
  principal="client/admin@ADS.MYKERBEROS.LOCAL";
```

Установите права доступа:

```
sudo chmod 777 /etc/kafka-connect-ads/conf/jass.conf
sudo chmod 777 /etc/kafka-connect-ads/conf/client_config.conf
```

Отредактируйте файл /etc/kafka-connect-ads/conf/connect-distributed.properties:

```
bootstrap.servers=vlan1.mykerberos.mydomain.ru:9092
security.protocol=SASL_PLAINTEXT
sasl.kerberos.service.name=kafka
sasl.jaas.config=com.sun.security.auth.module.Krb5LoginModule required \
  useKeyTab=true \
  storeKey=true \
  keyTab="/var/kerberos/krb5kdc/client.keytab" \
  principal="client/admin@ADS.MYKERBEROS.LOCAL";
```

Отредактируйте файл /etc/kafka-connect-ads/conf/connect-standalone.properties:

```
bootstrap.servers=vlan1.mykerberos.mydomain.ru:9092
security.protocol=SASL_PLAINTEXT
sasl.kerberos.service.name=kafka
sasl.jaas.config=com.sun.security.auth.module.Krb5LoginModule required \
  useKeyTab=true \
  storeKey=true \
  keyTab="/var/kerberos/krb5kdc/client.keytab" \
  principal="client/admin@ADS.MYKERBEROS.LOCAL";
```

Отредактируйте файл /etc/kafka-connect-ads/conf/connect-distributed.properties:

```
bootstrap.servers=vlan1.mykerberos.mydomain.ru:9092
security.protocol=SASL_PLAINTEXT
sasl.kerberos.service.name=kafka
sasl.jaas.config=com.sun.security.auth.module.Krb5LoginModule required \
  useKeyTab=true \
  storeKey=true \
  keyTab="/var/kerberos/krb5kdc/client.keytab" \
  principal="client/admin@ADS.MYKERBEROS.LOCAL";
```

Отредактируйте файл /etc/kafka-connect-ads/conf/connect-standalone.properties:

```
bootstrap.servers=vlan1.mykerberos.mydomain.ru:9092
security.protocol=SASL_PLAINTEXT
sasl.kerberos.service.name=kafka
sasl.jaas.config=com.sun.security.auth.module.Krb5LoginModule required \
  useKeyTab=true \
  storeKey=true \
  keyTab="/var/kerberos/krb5kdc/client.keytab" \
  principal="client/admin@ADS.MYKERBEROS.LOCAL";
```

Отредактируйте файл /etc/kafka-connect-ads/conf/connect-log4j.properties:

```
log4j.rootLogger=INFO, stdout, connectAppender

log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=[%d] %p %m (%c:%L)%n

log4j.appender.connectAppender=org.apache.log4j.DailyRollingFileAppender
log4j.appender.connectAppender.DatePattern='.'yyyy-MM-dd-HH
log4j.appender.connectAppender.File=${kafka.logs.dir}/connect.log
log4j.appender.connectAppender.layout=org.apache.log4j.PatternLayout
log4j.appender.connectAppender.layout.ConversionPattern=[%d] %p %m (%c:%L)%n

log4j.logger.org.reflections=ERROR
```

Kerberos с MS Active Directory

Ольга Семме

Содержание

Назад к содержанию

Подготовка к запуску Active Directory в кластере ADS

Запуск Active Directory в кластере ADS в интерфейсе ADCM

Основные понятия, используемые в Kerberos LDAP:

- **Active Directory** — это база данных и набор служб, которые соединяют пользователей с необходимыми сетевыми ресурсами.
- **LDAP-сервер** — иерархическая база данных, служба каталогов, основанная на Active Directory, применяется для централизованного хранения учетных записей. Учетные данные включают в себя принципалы, по которым происходит проверка валидности пользователя в Kerberos.
- **Легковесный протокол доступа к каталогам** (**Lightweight Directory Access Protocol, LDAP**) — протокол прикладного уровня для доступа к службе каталогов Active Directory, используется при керберизации кластера для заведения принципалов пользователей.
- **LDAPS** — активация LDAP с применением протокола SSL/TLS для создания безопасного трафика. Создание паролей для пользователей ADS с учетными записями в Active Directory при керберизации кластера возможно только с применением LDAPS.
- **Центр распределения ключей** (**Key Distribution Center, KDC**) — компонент системы управления доступом, отвечающий за обслуживание запросов пользователей на доступ к ресурсам путем предоставления тикетов доступа и сеансовых ключей. Он использует Active Directory в качестве базы данных учетных записей.
- **Distinguished Name, DN** — учетная запись в Active Directory. DN должно быть уникальным в пределах дерева. В DN описывается содержимое атрибутов в дереве (путь навигации) для доступа к конкретной записи.
  DN состоит из серии **относительных отличительных имен** (**Relative Distinguished Names, RDN**), определяемых путем перемещения вверх по дереву в направлении его корневой записи. RDN записываются слева направо.

Пример DN для пользователя, используемый в данной статье:

cn=admin-kafka,ou=kerberos,ou=adfs,dc=ad,dc=ranger-test.

В этой записи назначены следующие RDN:

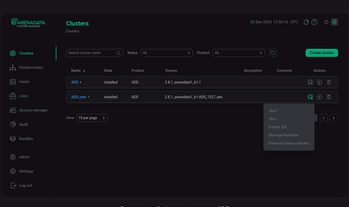
- cn — общее имя пользователя (common name).
- ou — организационная единица (organizational unit), контейнер в домене Active Directory, который может содержать пользователей, группы и компьютеры. Организационная единица может иметь несколько ou внутри себя.
- dc — части имени домена (domain component), представляют вершину дерева LDAP, которое использует DNS для определения своего пространства имен. Обозначение для домена Active Directory с DNS-именем `AD.RANGER-TEST` имеет вид `dc=ad,dc=ranger-test`. Имя `AD.RANGER-TEST` также присваивается для области (realm), для которой проводится аутентификация.

Подготовка к запуску Active Directory в кластере ADS

1. Настройте сервер Active Directory (LDAP-сервер).
2. Создайте учетную запись пользователя для подключения кластера к LDAP-серверу по протоколу LDAP.
3. Создайте сертификат для пользователя, активировав протокол LDAP через протокол SSL с использованием стороннего центра сертификации.
4. Установите пароль для учетной записи.
5. Привяжите к учетной записи имя принципала пользователя, например, `admin-kafka@AD.RANGER-TEST` и сформируйте файл *keytab* с этими принципалом и ключами.

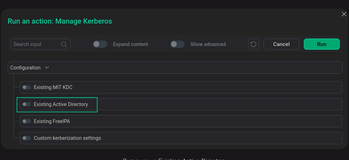
Запуск Active Directory в кластере ADS в интерфейсе ADCM

1. Убедитесь, что известны необходимые данные для запуска Active Directory:
   - **KDC hosts** — адрес хоста, где установлен Key Distribution Center, для керберизации с применением LDAP этот адрес совпадает с адресом LDAP-сервера;
   - **Realm** — обозначение области для аутентификации в Kerberos;
   - **Kadmin server** — сервер для установки системы администрирования Kerberos VS, для керберизации с применением LDAP этот адрес совпадает с адресом LDAP-сервера;
   - **Kadmin principal** — обозначение принципала пользователя, созданного для подключения кластера к Active Directory;
   - **Kadmin password** — пароль для принципала пользователя, созданного для подключения кластера к Active Directory;
   - **Admin DN** — учетная запись в Active Directory для пользователя, используемого для подключения кластера к Active Directory;
   - **LDAP URL** — адрес LDAP-сервера;
   - **Container DN** — отличительное имя ou, которое станет расположением по умолчанию для созданных во время керберизации объектов кластера на уровне ниже пользователя, описанного в Admin DN;
   - **TLS CA certificate Path** — путь к созданному сертификату CA для пользователя Admin DN (допускается не указывать, тогда ADCM обратится к LDAP-серверу);
   - **TLS CA certificate (optional)** — номер созданного сертификата CA для пользователя Admin DN (допускается не указывать, тогда ADCM обратится к LDAP-серверу).
2. Инициируйте включение MS Active Directory для выбранного кластера. Для этого примените действие ***Manage Kerberos***, нажав на иконку в столбце ***Actions***.



Включение Kerberos в кластере ADS

3. Включите переключатель ***Existing Active Directory*** в открывшемся окне.

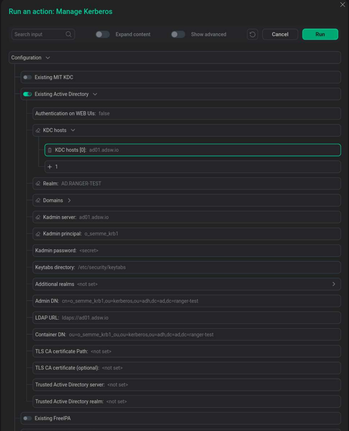


Включение Existing Active Directory

ПРИМЕЧАНИЕ

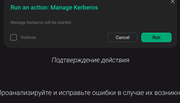
Включение ***Existing Active Directory*** можно совместить с опцией Custom kerberization settings.

4. Установите параметры конфигурации кластера ADS для MS Active Directory в соответствии со значениями, определенными для пользователя в каталоге LDAP, и нажмите ***Run***.



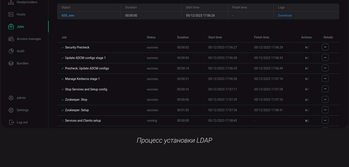
Настройка конфигурации кластера ADS для MS Active Directory

5. Подтвердите действие в открывшемся окне.



Подтверждение действия

6. Дождитесь завершения включения Kerberos LDAP. Проанализируйте и исправьте ошибки в случае их возникновения.



Процесс установки LDAP

ПРИМЕЧАНИЕ

Аутентификация по протоколу LDAP после запуска ***Manage Kerberos*** в сервисе Kafka осуществляется в соответствии со статьей Использование Kerberos с MS Active Directory в Kafka.

Содержание

Обзор
Запуск Kerberos Samba для кластера ADS в интерфейсе ADCM
Kerberos-параметры Samba

Обзор

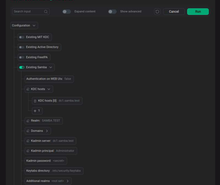
Samba поступает контролером домена Active Directory, поэтому настройка Kerberos аутентификации с Samba очень похожа на настройку аутентификации с ADS Active Directory.

Особенности CentOS 7

В CentOS 7 Samba1 не генерирует файлы автоматически, так как этот стандартный пакет Samba для CentOS 7 не поддерживает создание прокладок на удостоверяющем. Из-за такой конфигурации нам необходимо подготовить keytab-файлы вручную перед включением Kerberos Samba в ADS. Для этого вы можете изменить переменные в скрипте ниже и использовать его. В других ОС генерация происходит автоматически.

Запуск Kerberos Samba для кластера ADS в интерфейсе ADCM

1. Инициируйте включение Kerberos Samba для выбранного кластера. Для этого перейдите на кластерные действия **Manage Kerberos**, нажав на иконку ⋮ и выбрав **Actions**.
2. Включите переключатель *Existing Samba* в открывшемся окне.

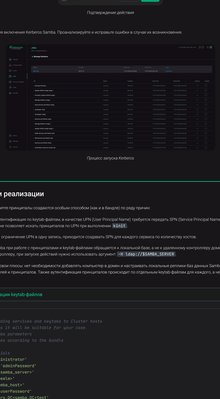


Включение Existing Samba

ПРИМЕЧАНИЕ

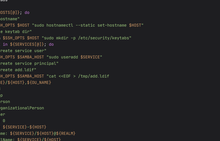
Включение **Existing Samba** можно совместить с опцией Custom kerberization settings.

3. Переходите в активное состояние переключатель **Show advanced**
4. Установите Kerberos-параметры Samba кластера ADS и нажмите **Run**.



Настройка конфигурации кластера ADS для Kerberos Samba

5. Подтвердите действие в открывшемся окне.



Подтверждение действия

6. Дождитесь завершения включения Kerberos Samba. Проконтролируйте и исправьте ошибки в случае их возникновения.

Процесс запуска Kerberos

Особенности реализации

В представленном скрипте специально создается особым способом (как и в бандле) по ряду причин:

- Чтобы работала аутентификация по keytab-файлам, в качестве UPN (User Principal Name) требуется передать SPN (Service Principal Name). Это необходимо из-за того, что Samba не может использовать произвольное UPN при выполнении `kinit`.
- Так как существуют ограничения UPN в одну запись, приходится создавать SPN для каждого сервиса по количеству хостов.
- По умолчанию Samba при работе с прокладками и keytab-файлами использует кэш в памяти, а не в удаленному контроллеру домена. Samba обращалась к контроллеру, при этом создавая дополнительную нагрузку. Поэтому мы отключили `DCACHE=FILE:/$SAMBA_DIR/krb5.kcache`.

У такого подхода есть свои плюсы: нет необходимости устанавливать клиент и администрировать локальные ветвления для данных Samba для создания серверных пользовательских прокладок. Также автоматически создаются keytab-файлы для каждого сервиса, а не по одному keytab-файлу на компьютер.

Скрипт для генерации keytab-файлов

```
#!/bin/bash
# Script for adding services and keytabs to Cluster hosts
# Edit: HOSTS as it will be suitable for your case
# Run: on Samba DC
# Split services according to the bundle

# Samba credentials
SAMBA_USER='Administrator'
SAMBA_PASSWORD='passwordvalue'
SAMBA_SERVICE='samba_server'
REALM='samba.realm'
SAMBA_HOST='samba_host'
USER_PASSWORD='user-password'
DJ_HOST='CN=Users,DC=samba,DC=test'

# Samba
smb.conf -s HOSTS

SSH_OPTS='-o GlobalKnownHostsFile=/dev/null -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no'

HOSTS=(
  "list your hosts here"
)
encoded_password=$(echo -n "${USER_PASSWORD}"| iconv -f UTF-8 -t UTF-16LE | base64)

# Samba
SERVICES=(HTTP nifi zookeeper kafka ksql-server kafka-rest schema-registry)

mkdir /tmp/keytabs

for HOST in ${HOSTS[@]}; do
  echo "Set hostname"
      ssh $SSH_OPTS $HOST "sudo hostnamectl --static set-hostname $HOST"
  echo "Create keytab dir"
      ssh $SSH_OPTS $HOST "sudo mkdir -p /etc/security/keytabs"
  for SERVICE in ${SERVICES[@]}; do
      echo "Create service user"
      ssh $SSH_OPTS $HOST "sudo useradd ${SERVICE}"
      echo "Create service principal"
      echo "Create and telf"
      ssh $SSH_OPTS $SAMBA_HOST "set +eEF -x /tmp/add_telf
DN: DN=${SERVICE}/${HOST},${DJ_NAME}
changetype: add
objectClass: top
objectClass: person
objectClass: organizationalPerson
objectClass: user
accountExpires: 0
servicePrincipalName: ${SERVICE}/${HOST}
userPrincipalName: ${SERVICE}/${HOST}@${REALM}
servicePrincipalName: ${SERVICE}/${HOST}
distinguishedName: CN=${SERVICE}/${HOST},${DJ_NAME}
userAccountControl: 514

DN: CN=${SERVICE}/${HOST},${DJ_NAME}
changetype: modify
replace: unicodePwd
unicodePwd:: $encoded_password

DN: CN=${SERVICE}/${HOST},${DJ_NAME}
changetype: modify
replace: userAccountControl
userAccountControl: 66048

EOF"
      echo "sudo ldbmodify -H ldap://$SAMBA_SERVER /tmp/add_telf -U $SAMBA_USER --password=$SAMBA_PASSWORD"
      ssh $SSH_OPTS $SAMBA_HOST "sudo ldbmodify -H ldap://$SAMBA_SERVER /tmp/add_telf -U $SAMBA_USER --
password=$SAMBA_PASSWORD"

      echo "Delete keytab"
      ssh $SSH_OPTS $SAMBA_HOST "sudo rm -f /tmp/${SERVICE.service.keytab"

      echo "Extract keytab"
      TMP_SCRIPT="/tmp/extract_keytab_${SERVICE}_${HOST}.sh"
      KTUTIL_SCRIPT="/tmp/ktutil_commands_${SERVICE}_${HOST}.txt"
      cat <<EOT > $TMP_SCRIPT
#!/bin/bash
rm -f /tmp
sudo ktutil -k $KTUTIL_SCRIPT
EOT

      cat <<EOT > $KTUTIL_SCRIPT
add_entry -password -p ${SERVICE}/${HOST}@${REALM} -k 1 -e aes256-cts-hmac-sha1-96
${USER_PASSWORD}
add_entry -password -p ${SERVICE}/${HOST}@${REALM} -k 1 -e aes128-cts-hmac-sha1-96
${USER_PASSWORD}
add_entry -password -p ${SERVICE}/${HOST}@${REALM} -k 1 -e arcfour-hmac
${USER_PASSWORD}
wkt ${SERVICE}.service.keytab
EOT
      scp $SSH_OPTS $TMP_SCRIPT $SAMBA_HOST:$TMP_SCRIPT
      scp $SSH_OPTS $KTUTIL_SCRIPT $SAMBA_HOST:$KTUTIL_SCRIPT
      ssh $SSH_OPTS $SAMBA_HOST "chmod +x $TMP_SCRIPT && $TMP_SCRIPT && rm $TMP_SCRIPT && rm $KTUTIL_SCRIPT"
      rm $TMP_SCRIPT
      rm $KTUTIL_SCRIPT

      echo "Try split"
      ssh $SSH_OPTS $SAMBA_HOST "sudo klist -k /tmp/${SERVICE.service.keytab"
      echo "Change permission"
      ssh $SSH_OPTS $SAMBA_HOST "sudo chmod 777 /tmp/${SERVICE}.service.keytab"
      echo "Input keytab to local"
      scp $SAMBA_HOST:/tmp/${SERVICE}.service.keytab /tmp/keytabs/${SERVICE}.service.keytab
      echo "Input keytab to host"
      scp /tmp/keytabs/${SERVICE}.service.keytab $HOST:/tmp/${SERVICE}.service.keytab
      ssh $SSH_OPTS $HOST "sudo mv /tmp/${SERVICE}.service.keytab /etc/security/keytabs/${SERVICE}.service.keytab"
      echo "Clean up local temporary keytab"
      rm /tmp/keytabs/${SERVICE}.service.keytab
  done

  echo "Change owner and group"


  ssh $SSH_OPTS $HOST sudo chown zookeeper:zookeeper /etc/security/keytabs/zookeeper*
  ssh $SSH_OPTS $HOST sudo chown nifi:nifi /etc/security/keytabs/nifi*
  ssh $SSH_OPTS $HOST sudo chown nifi:nifi /etc/security/keytabs/nifi*
  ssh $SSH_OPTS $HOST sudo chown kafka:kafka /etc/security/keytabs/kafka*
  ssh $SSH_OPTS $HOST sudo chown ksql:kafka /etc/security/keytabs/ksql-server*
  ssh $SSH_OPTS $HOST sudo chown kafka:kafka /etc/security/keytabs/kafka-rest*
  ssh $SSH_OPTS $HOST sudo chown schema-registry:kafka /etc/security/keytabs/schema-registry*

  echo "Change permissions"

  ssh $SSH_OPTS $HOST "sudo chmod -R 777 /etc/security/keytabs/*"

done
```

Для включения Kerberos Samba на кластах с CentOS 7, где используются дефолтные слабые алгоритмы шифрования и придерживаться не была настроена среда Kerberos, включение аутентификации Samba выполняется в несколько этапов:

1. Перед запуском скрипта убедитесь, что служба доменных имен (DNS) локального компьютера (хостов), с которого запущен скрипт, а также хостов кластера ADS настроены для подключения по SSH к серверу Samba.
2. При первом включении **Manage Kerberos** активируйте опцию **Custom kerberization settings**, в которой установите значение `true`. Только для параметров **Set up Kadmin ads + Custom kerberization on hosts** `true` и **Use Active Sync** установлено или отмечено и создано или конфигурационные файлы Kerberos.
3. На странице кластера Primary configuration в группе параметров **Kerberos** проверьте параметр **Custom krb5.conf** значение `true`. Далее откройте шаблон пользовательского krb5.conf и отредактируйте опцию **Custom krb5.conf** и сохраните изменения (`[libdefaults]` как в примере krb5.conf, приведенном ниже, и сохраните конфигурацию).
4. При первом включении **Manage Kerberos** установите и опцию **Custom kerberization settings** значение `true` для всех параметров, кроме параметров **Set up principals and keytabs**.

Пример пользовательского krb5.conf

```
[logging]
 default = FILE:/var/log/krb5libs.log
 kdc = FILE:/var/log/krb5kdc.log
 admin_server = FILE:/var/log/kadmind.log

[libdefaults]
 dns_lookup_realm = false
 ticket_lifetime = 24h
 # renew_lifetime = 7d
 forwardable = true
 rdns = false
 pkinit_anchors = /etc/pki/tls/certs/ca-bundle.crt
 default_realm = SAMBA.TEST
 permitted_enctypes = aes256-cts-hmac-sha1-96 aes128-cts-hmac-sha1-96 arcfour-hmac rc4-hmac arcfour-hmac-md5
 default_tkt_enctypes = aes256-cts-hmac-sha1-96 aes128-cts-hmac-sha1-96 rc4-hmac
 default_tgs_enctypes = aes256-cts-hmac-sha1-96 aes128-cts-hmac-sha1-96 rc4-hmac
 allow_weak_crypto = true

[realms]
 SAMBA.TEST = {
  admin_server = dc1.samba.test
  kdc = dc1.samba.test
 }
```

Kerberos-параметры Samba

Параметр	Описание
Authentication on WEB UIs	Включает аутентификацию Kerberos для веб-интерфейсов
KDC hosts	Один или несколько хостов контроллера домена
Realm	Kerberos realm – сеть, содержащая KDC-хосты и множество клиентов
Domains	Домены, в которых находятся хосты
Kadmin server	Адрес хоста, на котором запущен `kadmin`
Kadmin principal	Имя принципала для подключения через `kadmin`, например `admin@MY-CENTRAL3.INTERNAL`
Kadmin password	Пароль принципала для подключения через `kadmin`
Keytabs directory	Директория keytab-файла, содержащего записи одного или нескольких принципалов и их ключи
Additional realms	Дополнительные сети Kerberos realm
LDAP URL	LDAP URL состоит из префикса `ldap://` или `ldaps://`, имени хоста или IP-адреса и порта LDAP-сервера
Container DN	Отличительное имя контейнера
Trusted Active Directory server	Сервер с DC для настраиваемой доверенности
Trusted Active Directory realm	Область доверия между realm

ПРИМЕЧАНИЕ

Аутентификация Kerberos на основе Samba после запуска **Manage Kerberos** с сервера Kafka осуществляется в соответствии со статьей «Использование Kerberos с ADS Active Directory и Kafka».

Custom kerberization settings

Ольга Семме

Содержание

Назад к содержанию

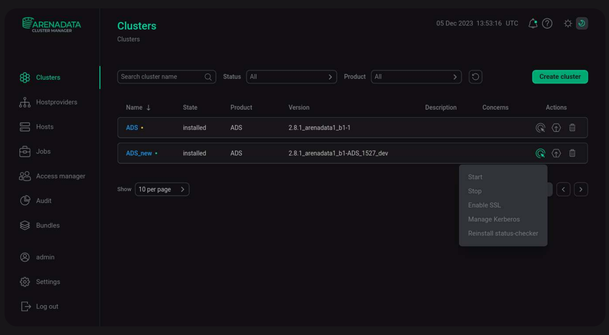
Рекомендации для пользовательских keytab-файлов

Опция ***Custom kerberization settings*** позволяет пользователю выбирать шаги керберизации, например, создание принципалов и keytab-файлов.

Каждый протокол, доступный при выборе действия кластерного действия ***Manage Kerberos*** (MIT Kerberos, MS Active Directory, FreeIPA) можно совместить с опцией ***Custom kerberization settings***.

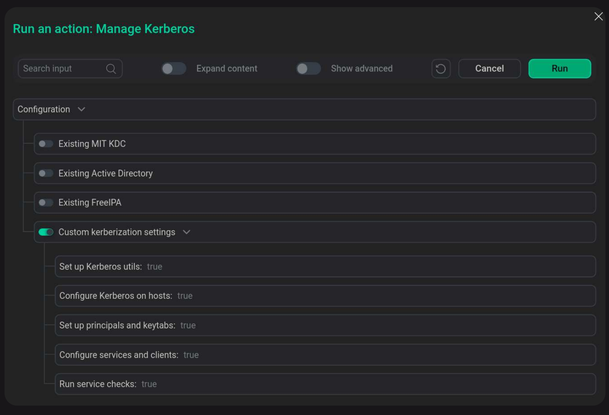
Для установки опции выполните шаги:

1. Инициируйте включение Kerberos для выбранного кластера. Для этого примените действие ***Manage Kerberos***, нажав на иконку в столбце ***Actions***.



Включение Kerberos в кластере ADS

2. Включите переключатель ***Custom kerberization settings*** в открывшемся окне, раскройте дерево параметров, установите необходимые параметры и нажмите ***Run***.



Параметры Custom kerberization settings

Описание параметров Custom kerberization settings

Параметр	Описание	Значение по умолчанию
Set up Kerberos utils	Активирует возможность устанавливать и удалять клиенты и утилиты Kerberos. Данный параметр учитывается в действиях ***Expand*** и ***Install***	True
Configure Kerberos on hosts	Активирует возможность конфигурировать кластер, включая *krb5.conf*, *ldap.conf*	True
Set up principals and keytabs	Активирует возможность создавать, пересоздавать и удалять принципалов и keytab-файлы. Пароли для принципалов генерируются случайным образом перед созданием keytab-файлов. Данный параметр учитывается в действиях ***Expand*** и ***Install***. Бандл ADCM установит owner и разрешения для keytab-файлов только если этот флаг установлен. В случае отсутствия прав администратора пользователь должен предоставить подготовленный keytab-файл с правильно установленными owner и разрешениями (см. Рекомендации для пользовательских keytab-файлов)	True
Configure services and clients	Установка параметра не влияет на работу ADS. Пользовательские настройки выполняются при помощи шаблона JAAS-файла	True
Run service checks	Активирует возможность проводить проверку сервисов	True

Рекомендации для пользовательских keytab-файлов

Ниже представлены рекомендации для значений owner, group и прав для keytab-файлов.

Рекомендации для keytab-файлов

Short name компонента	Owner keytab-файла	Group keytab-файла	Права
kafka	kafka	kafka	600
kafka-rest	kafka-rest	kafka	600
ksql-server	ksql	kafka	600
nifi	nifi	nifi	600
nifi-registry	nifi	nifi	600
schema-registry	schema-registry	kafka	600
kafka-connect	kafka-connect	kafka	600
zookeeper	zookeeper	zookeeper	600
HTTP	nifi	nifi	640

SASL PLAINTEXT

Ольга Семме

Назад к содержанию

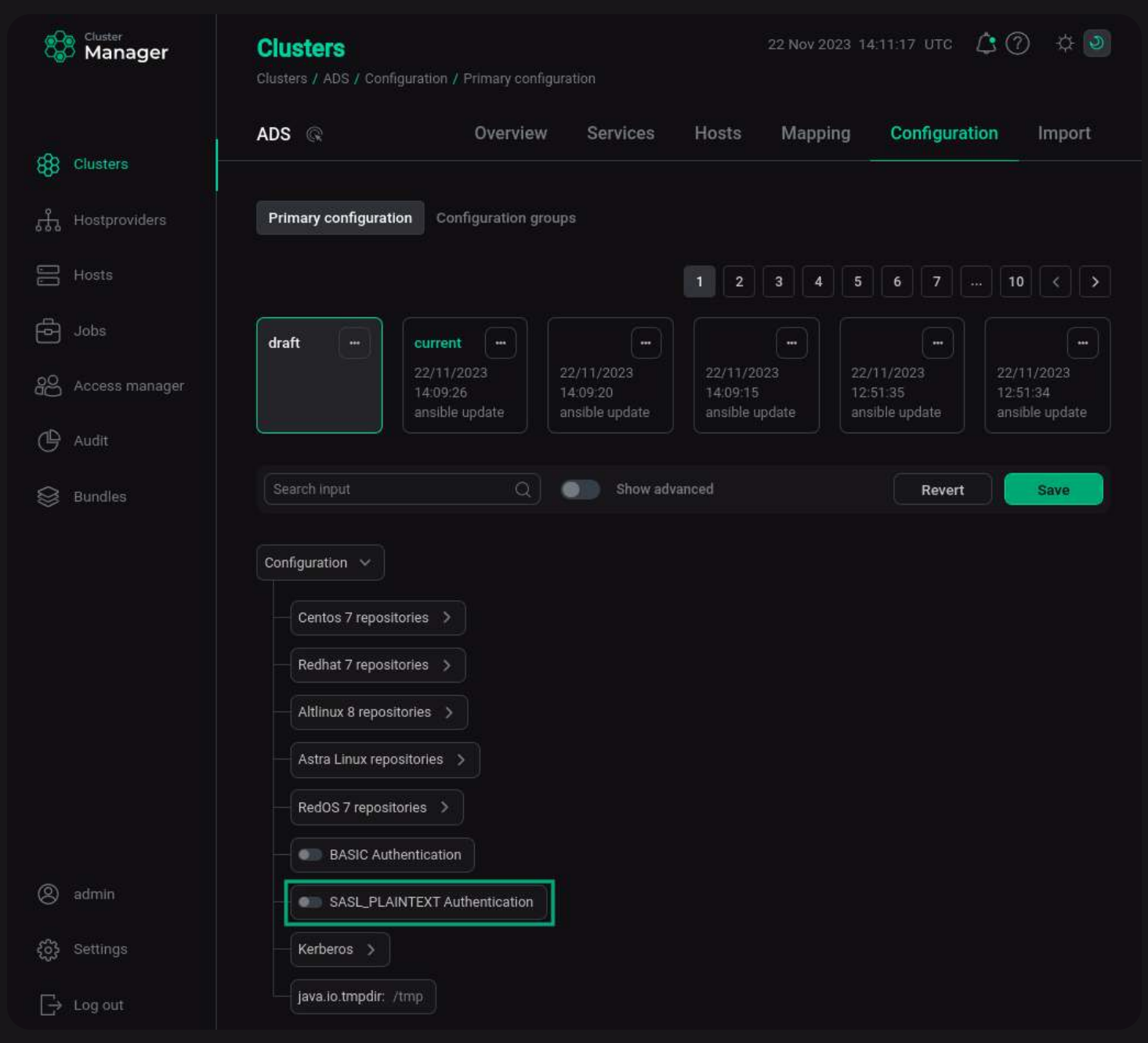
SASL — фреймворк для аутентификации и обеспечения безопасности данных, используемый в интернет-протоколах.

PLAIN — простой механизм передачи паролей открытым текстом.

PLAINTEXT — аутентификатор, который конфигурируется для поддержки аутентификационного механизма PLAIN.

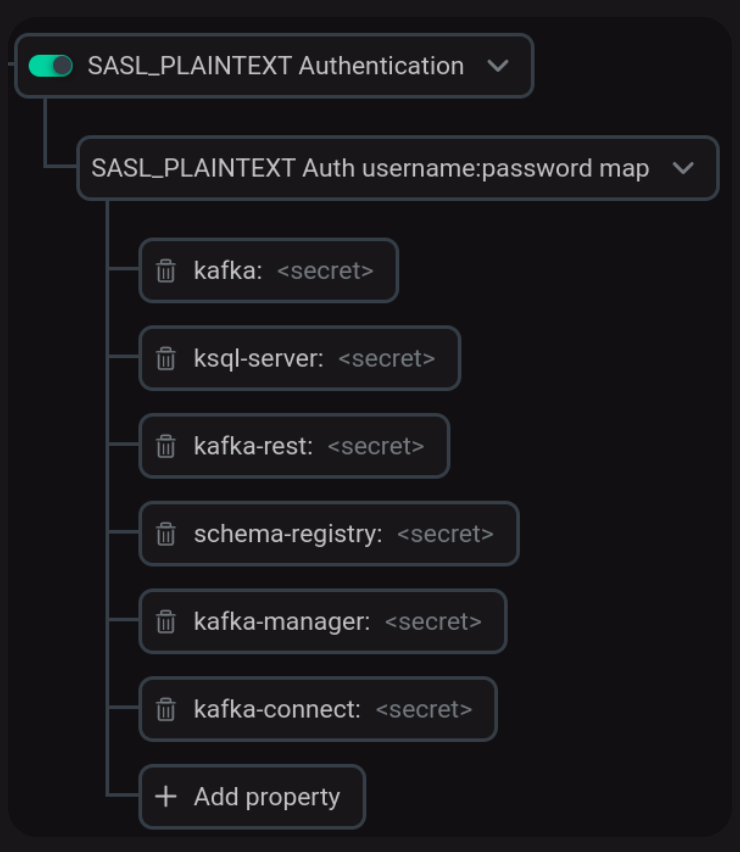
Для включения аутентификации SASL PLAINTEXT с помощью ADCM выполните шаги:

1. На вкладке конфигурирования кластера в дереве конфигурационных настроек включите переключатель ***SASL_PLAINTEXT Authentication*** и раскройте узел ***SASL_PLAINTEXT Auth username:password map***.



Включение SASL_PLAINTEXT Authentication в кластере ADS

В открывшихся узлах находятся установленные по умолчанию зашифрованные пароли пользователя для каждого сервиса, предназначенного для работы с Kafka.



Узлы конфигурации с пользователями и паролями SASL_PLAINTEXT Authentication

При первом включении ***SASL_PLAINTEXT Authentication*** автоматически формируются имена и пароли пользователей каждого сервиса:

```
kafka: kafka-password
ksql-server: ksql-server-password
kafka-rest: kafka-rest-password
schema-registry: schema-registry-password
kafka-connect: kafka-connect-password
```

Пароли по умолчанию рекомендуется изменить.

2. Для изменения пароля нажмите на ***<secret>*** рядом с именем пользователя, введите новый пароль, подтвердите пароль и нажмите ***Apply***.

3. Сохраните конфигурацию, нажав ***Save***.

4. Перезагрузите все сервисы, для которых была настроена аутентификация. Для этого примените действие Restart, нажав на иконку в столбце ***Actions***.

ПРИМЕЧАНИЕ

После включения аутентификации SASL PLAINTEXT работа пользователя в сервисе Kafka осуществляется в соответствии со статьей Использование SASL PLAINTEXT в Kafka.

Защита канала по протоколу SSL

Ольга Семко

Содержание

Назад к содержанию

- Предварительные требования для включения SSL
- Создание сертификата SSL для Certification Authority
  - Последовательность создания сертификата
  - Создание сертификата SSL для Certification Authority при помощи скрипта
- Включение SSL в кластере ADS при помощи ADCM
- Выключение SSL в кластере ADS при помощи ADCM

В данной статье описаны действия для включения защиты канала по протоколу SSL при аутентификации в ADS.

Предварительные требования для включения SSL

- На каждом хосте кластера должны размещаться хранилище ключей и сертификатов keystore.jks (уникальное для каждого хоста кластера), должен быть известен путь к нему и его пароль.
- На каждом хосте кластера должно размещаться хранилище ключей и сертификатов truststore.jks (общее для всех хостов кластера), должен быть известен путь к нему и его пароль.
- Должен быть известен список DN всех хостов кластера (вида `CN=ads-host-1.example.com, OU=AD, O=AD, L=MSK, ST=RU, C=RU`), сформированных при создании сертификатов SSL.
- Должны быть известны имя пользователя, имеющего связь в каталоге Active Directory, которой будет выполнять функции администратора NiFi, и его пароль.

Создание сертификата SSL для Certification Authority

Последовательность создания сертификата

1. Сгенерируйте пару ключей по алгоритму RSA для каждого хоста.
2. Создайте на каждом хосте хранилище ключей keystore.jks.
3. Создайте SSL-сертификат для каждого хоста.
4. Сохраните все сертификаты в одном месте.
5. Скопируйте все сертификаты на каждый хост в хранилище keystore.jks.
6. Подпишите сертификаты ключом, предназначенным центру по выпуску сертификатов, включенных в хранилище сертификатов Java.
7. Создайте хранилище truststore.jks, содержащее необходимые сертификаты.
8. Импортируйте truststore.jks в Java CA store.
9. Создайте и импортируйте сертификат OpenSSL для Nginx.

ВНИМАНИЕ

Для корректного включения аутентификации в сервисе NiFi при создании сертификатов необходимо указать и заполнить поле `SAN` (SubjectAltName) при этом опция `-ext` сохранена, для самостоятельных сертификатов, заполненное поле выглядит так: `-ext "SAN=DNS:$HOST"`.

Создание сертификата SSL для Certification Authority при помощи скрипта

Для тестовых и стендовых целей при создании сертификатов для каждого хоста и импорта сертификатов в хранилище ключей Java можно использовать следующий скрипт generate.sh.

При помощи данного скрипта возможно как создание сертификатов, так и перезапускать уже созданные.

Запуск скрипта выполняется на хосте инсталлера (при доступе к хостам кластера по FQDN и внутренних IP

Ниже приведена последовательность создания скрипта generate.sh

1. Введите команду для создания файла скрипта

```
$ sudo vim /tmp/ssl/generate.sh
```

2. Заменить для следующих строк скрипта (только для vim) следующие значения на значения, которые относятся к вашему случаю:

- `NUMHOSTS=` - в этой строке укажите значение (буквы или цифры), индивидуальное для каждого хоста. Значение `{1 2 3}` приведены как пример для списка хостов.
  - sce-ads-test-1.ru-central1.internal;
  - sce-ads-test-2.ru-central1.internal;
  - sce-ads-test-3.ru-central1.internal;
- `HOSTSC` - в этой строке укажите общую часть FQDN хоста, заменив `$i` на значения (буквы или цифры), указанные в параметре `NUMHOSTS`. Значение `sce-ads-test-$i.ru-central1.internal` приведено как пример.
- `SIGNPW` - пароль для хранилищ ключей, рекомендуется заменить.

3. Заполните generate.sh текстом скрипта, приведенным ниже, с учетом индивидуальных параметров.

generate.sh

```
#!/bin/bash

# Script that is used to generate and import self-signed certificates to Java keystore
# Edit NUMHOSTS and HOSTS so it will be suitable for your case

declare -a NUMHOSTS
declare -a HOSTS

SSH_OPTS="-o UserKnownHostsFile=/dev/null -o UserKnownHostsFile=/dev/null -o StrictHostKeyChecking=no"
NUMHOSTS=(1 2 3)
HOSTS=($(for i in ${NUMHOSTS[@]}; do echo "sce-ads-test-$i.ru-central1.internal"; done))

for HOST in $HOSTS; do
    echo "update-ca-trust"
    ssh $SSH_OPTS $HOST "sudo update-ca-trust extract"
done

echo Generate keystore jks on each host
for HOST in $HOSTS; do
    echo "remove old alias"
    ssh $SSH_OPTS $HOST "keytool -delete -alias $HOST -keystore /tmp/keystore_jks -storepass bigdata"

    echo "Generating keypair"
    ssh $SSH_OPTS $HOST "keytool
                        -genkey \
                        -noprompt \
                        -keyalg RSA \
                        -ext san=dns:$HOST \
                        -alias $HOST \
                        -dname \"CN=$HOST, OU=AD, O=AD, L=MSK, S=RU, C=RU\" \
                        -keystore /tmp/keystore_jks \
                        -storepass bigdata \
                        -keypass bigdata \
                        -validity 540 \
                        -keysize 2048"
done

echo
echo Export certificates
for HOST in $HOSTS;do
        ssh $SSH_OPTS $HOST "keytool
                            -export \
                            -file /tmp/$HOST.crt \
                            -keystore /tmp/keystore_jks \
                            -storepass bigdata \
                            -alias $HOST \
                            -rfc";
done

echo
echo Collect all certificates
for HOST in $HOSTS; do
        scp $SSH_OPTS $HOST:/tmp/$HOST.crt /tmp/

#       scp $SSH_OPTS $HOST:/tmp/*.crt /tmp/
done

echo
echo Transfer certificates on hosts
for HOST in $HOSTS; do
        for CERT in $HOSTS;do
            scp $SSH_OPTS /tmp/$CERT.crt $HOST:/tmp/
        done

#       scp $SSH_OPTS /tmp/*.crt $HOST:/tmp/
done

echo
echo Import certificates on each host
for HOST in $HOSTS; do

        ssh $SSH_OPTS $HOST "for CERT in $(echo ${HOSTS[*]}); do

                            echo \"remove old alias before import\"
                            keytool -delete -noprompt -alias \$CERT -keystore /tmp/truststore_jks -storepass bigdata

                            echo \"Import Cert\"
                            keytool \
                                    -import \
                                    -noprompt \
                                    -alias \$CERT \
                                    -file /tmp/\$CERT.crt \
                                    -keystore /tmp/truststore_jks \
                                    -storepass bigdata;
                            sudo bash -c \"cat /tmp/\$CERT.crt >> /etc/pki/ca-trust/source/ca-bundle.crt\";
                            done";
done

echo
echo Import truststore to Java CA store
for HOST in $HOSTS; do
        ssh $SSH_OPTS $HOST "sudo keytool       -importkeystore \
                                                -noprompt \
                                                -srckeystore /tmp/truststore_jks \
                                                -destkeystore /etc/pki/java/cacerts \
                                                -deststorepass changeit \
                                                -srcstorepass bigdata"

done

echo
echo Create directories for certs
for HOST in $HOSTS; do
        ssh $SSH_OPTS $HOST "[ -d /etc/ssl ] || sudo mkdir /etc/ssl;
                            [ -d /etc/ssl/certs ] || sudo mkdir /etc/ssl/certs"
done

echo
echo Create and import cert for Nginx
for HOST in $HOSTS; do
        ssh $SSH_OPTS $HOST "sudo openssl req -newkey rsa:4096 -days 365 -nodes -x509 -subj '/C=RU/ST=Moscow/L=Moscow/O=Arenadata/OU=ADS/CN=$HOST' -keyout /etc/ssl/host_cert.key -out /etc/ssl/certs/host_cert.crt";
        ssh $SSH_OPTS $HOST "sudo bash -c \"cat /etc/ssl/certs/host_cert.crt >> /etc/pki/ca-trust/source/ca-bundle.crt\""
done
```

4. Дайте разрешения на выполнение скрипта:

```
$ sudo chmod +x /tmp/ssl/generate.sh
```

5. Запустите скрипт:

```
$ /tmp/ssl/generate.sh
```

6. Скопируйте полученные.
7. Перезапуск на каждом хосте нужные хранилищ и файлов:
   - /etc/pki/host_cert.key
   - /tmp/truststore.jks
   - /tmp/keystore.jks

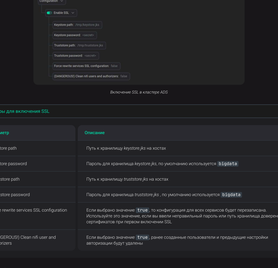
ПРИМЕЧАНИЕ

Для сервиса NiFi необходимо выполнить дополнительную конфигурацию в соответствии со статьей Аутентификация в сервисе NiFi.

Включение SSL в кластере ADS при помощи ADCM

Для включения SSL выполните следующие шаги:

1. Примените кластерные действие **Manage SSL**, нажав на иконку ⚙ в столбце **Actions** кластера ADS.
2. В открывшемся окне активируйте переключатель **Enable SSL**, раскройте дерево конфигураций и введите необходимые значения параметров.



Включение SSL в кластере ADS

Параметры для включения SSL

Параметр	Описание
Keystore path	Путь к хранилищу keystore.jks на хостах
Keystore password	Пароль для хранилищ keystore.jks, по умолчанию используется `bigdata`
Truststore path	Путь к хранилищу truststore.jks на хостах
Truststore password	Пароль для хранилищ truststore.jks , по умолчанию используется `bigdata`
Force rewrite services SSL configuration	Если выбрано значение `true`, то конфигурация для всех сервисов будет перезаписана используемыми настройками, если вы ранее неправильный пароль или путь хранилища доменных сертификатов при первом включении SSL
[DANGEROUS] Clean nifi users and authorizers	Если выбрано значение `true`, ранее созданные пользователи и продуктовые настройки авторизации будут удалены

3. Нажмите **Run**.
4. Подтвердите действие в открывшемся окне:

Подтверждение действия

5. Дождитесь завершения включения SSL. Проанализируйте и исправьте ошибки в случае их возникновения.

Процесс запуска SSL

ПРИМЕЧАНИЕ

Работа в сервисах после включения SSL осуществляется в соответствии со статьями:

- Защита канала по протоколу SSL в Kafka
- Аутентификация в сервисе NiFi

Выключение SSL в кластере ADS при помощи ADCM

Для выключения SSL выполните следующие шаги:

1. Примените кластерное действие **Manage SSL**, нажав на иконку ⚙ в столбце **Actions** кластера ADS.
2. В открывшемся окне деактивируйте переключатель **Enable SSL**.

Выключение SSL в кластере ADS

3. Нажмите **Run**.
4. Подтвердите действие в открывшемся окне:

Подтверждение действия

5. Дождитесь завершения выключения SSL. Проанализируйте и исправьте ошибки в случае их возникновения.

Обзор Arenadata Platform Security

Ольга Семме

Назад к содержанию

Arenadata Platform Security (ADPS) — это специализированный фреймворк для управления безопасностью, построенный на базе сервисов Apache Ranger и Apache Knox. ADPS может использоваться в ADS для обеспечения авторизации и управления доступом к сервисам на основе преднастроенных политик.

Перед установкой ADPS необходимо убедиться, что выполнены следующие требования:

- Кластер Arenadata Streaming версии 1.6.2 или выше установлен и настроен согласно руководству Установка Arenadata Streaming.
- Соблюдены все требования к установке ADPS:
  - Требования к оборудованию
  - Требования к сети
  - Программные требования
- Сервисы ADPS не должны использовать хосты кластера ADS. Рекомендуется установка ADPS на отдельной виртуальной машине или выделенном сервере.
- При использовании внешней СУБД для хранения метаданных Ranger (PostgreSQL/MySQL) она должна быть запущена и доступна. При установке Ranger создаются два новых пользователя (имена по умолчанию: `rangeradmin` и `rangerlogger`) и две новые базы данных (имена по умолчанию: `ranger` и `ranger_audit`).
- Необходимо настроить один из следующих типов аутентификации:
  - SASL PLAINTEXT
  - MIT Kerberos
  - MS Active Directory

ПРИМЕЧАНИЕ

Подробное описание установки ADPS приведено в статьях руководства Установка Arenadata Platform Security.

После установки ADPS и его интеграции с кластером ADS можно перейти к работе с плагинами Ranger, доступными для сервисов ADS.

Активация плагина Ranger и использование Ranger в сервисах ADS описаны в статьях:

- для сервиса Kafka:
  - Плагин Ranger Kafka
  - Использование Ranger в Kafka
- для сервиса NiFi:
  - Плагин Ranger NiFi
  - Использование Ranger в NiFi

Интеграция с кластером ADS

Ольга Семме

Содержание

Назад к содержанию

Интеграция кластера ADPS с кластером ADS

Доступ к веб-интерфейсу Ranger

После проверки аутентификации пользователя необходимо определить его права доступа. Права доступа пользователя к ресурсам определяются авторизацией.

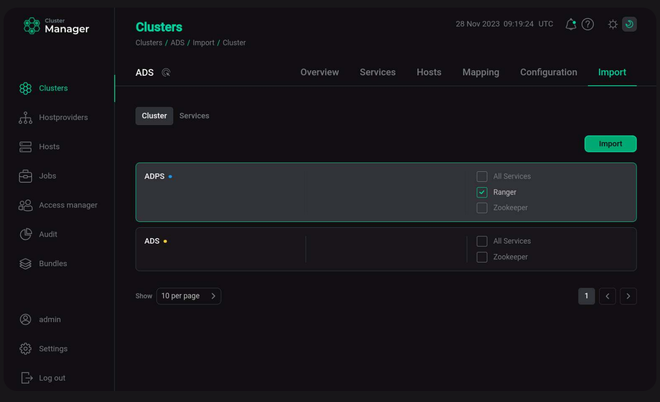
Возможности Ranger:

- Создание служб для сервиса Kafka ADS и добавление политики доступа пользователей к службам.
- Создание служб на основе тегов — позволяет управлять доступом к ресурсам нескольких компонентов без создания отдельных служб и политик в каждом компоненте.
- Использование Ranger Tab Sync для синхронизации хранилища тегов Ranger с внешней службой метаданных, например Apache Atlas.

Интеграция кластера ADPS с кластером ADS

Для включения интеграции кластера ADPS с кластером ADS:

1. Перейдите во вкладку ***Import*** кластера ADS.
2. Выберите сервис ***Ranger*** ранее добавленного кластера ADPS.
3. Кликните ***Import***.

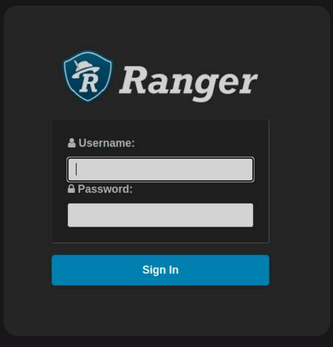


Включение Import Ranger ADPS

Доступ к веб-интерфейсу Ranger

Для доступа к веб-интерфейсу Ranger:

1. Введите в браузере обозначение хоста, на котором установлен ADPS, в качестве номера порта используйте значение, указанное в параметре ***ranger.service.http.port*** (по умолчанию 6080) сервиса Ranger.
2. Введите учетные данные администратора и кликните ***Sign in*** для авторизации в качестве администратора Ranger.

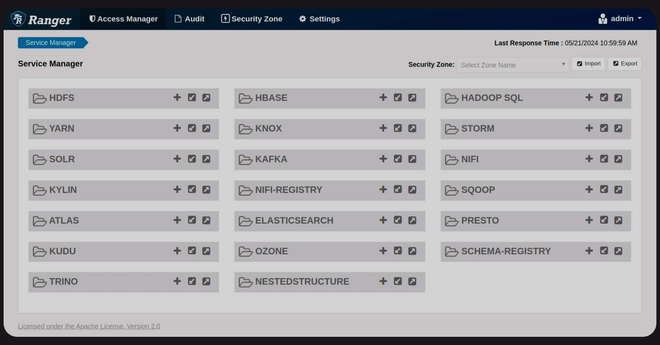


Вход в Ranger

ПРИМЕЧАНИЕ

Пароли для доступа к веб-интерфейсу Ranger задаются перед установкой сервисов кластера ADPS в разделе ***Credentials*** конфигурации сервиса Ranger.

В результате открывается окно интерфейса ***Service manager***.



Service manager Ranger

Метрики мониторинга кластера ADS

Ольга Семме

Содержание

Типы метрик

Просмотр метрик мониторинга

- Graphite
- Grafana

Назад к содержанию

В статье описываются метрики мониторинга кластера ADS. Для получения информации об установке мониторинга обратитесь к статьям:

- Online-установка
- Offline-установка

Типы метрик

Для ADS-кластера доступны две группы метрик: системные метрики и сервисные метрики.

В Graphite системные метрики отображаются в разделе ***System_metrics***, сервисные метрики — в разделе ***services***.

Системные метрики показывают общие характеристики хостов кластера, связанные, как правило, с потреблением ресурсов. Системные метрики приведены в таблице ниже.

System metrics

Группа метрик	Описание
cpu	Уровень загруженности процессора
diskspace	Заполнение диска
files	Файловая статистика
iostat	Производительность операций ввода/вывода
loadavg	Средние значения загрузки системы
memory	Использование памяти
netstat	Статистика сетевых подключений
network	Производительность сетевых интерфейсов
uptime	Время безотказной работы

Сервисные метрики показывают характеристики сервисов, доступные для хостов ADS-кластера.

Метрики для каждого сервиса приведены в таблицах ниже.

Kafka

Группа метрик	Название метрики	Описание
ReplicaManager	AtMinIsrPartitionCount	Количество партиций, соответствующих значению minIsr (`min.insync.replicas`)
	FailedIsrUpdatesPerSec	Количество неудачных обновлений синхронизированных реплик (In-Sync Replicas, ISR) в секунду. Метрика имеет следующие атрибуты: • количество событий; • скорость за 15 мин; • скорость за 5 мин; • средняя скорость; • скорость за 1 мин.
	IsrExpandsPerSec	Скорость увеличения числа синхронизированных реплик (In-Sync Replicas, ISR). Метрика имеет следующие атрибуты: • количество событий; • скорость за 15 мин; • скорость за 5 мин; • средняя скорость; • скорость за 1 мин.
	IsrShrinksPerSec	Скорость сокращения синхронизированных реплик (In-Sync Replicas, ISR). Метрика имеет следующие атрибуты: • количество событий; • скорость за 15 мин; • скорость за 5 мин; • средняя скорость; • скорость за 1 мин.
	LeaderCount	Количество лидеров партиций
	MaxLag	Максимальная задержка
	OfflineReplicaCount	Количество реплик в состоянии Offline
	PartitionCount	Количество партиций
	ReassigningPartitions	Количество переназначенных партиций
	UnderMinIsrPartitionCount	Количество партиций, число синхронизированных реплик (In-Sync Replicas, ISR) которых меньше minIsr
	UnderReplicatedPartitions	Количество не полностью реплицированных партиций (ISR < числа текущих реплик)
broker Метрики группы имеют следующие атрибуты: • количество событий; • скорость за 15 мин; • скорость за 5 мин; • средняя скорость; • скорость за 1 мин.	BytesInPerSec	Количество входных байтов в секунду
	BytesOutPerSec	Количество выходных байтов в секунду
	BytesRejectedPerSec	Скорость передачи отклоненных данных (байтов в секунду)
	FailedFetchRequestsPerSec	Количество запросов на получение, которые не удалось выполнить в секунду
	FailedProduceRequestsPerSec	Количество запросов производителя сообщений, которые не удалось выполнить в секунду
	FetchMessageConversionsPerSec	Количество преобразований получения сообщений в секунду
	InvalidMagicNumberRecordsPerSec	Количество ошибок в секунду при проверке сообщения из-за недопустимого числа Magic. Magic Byte Protocol используется для обратной совместимости формата сообщения
	InvalidMessageCrcRecordsPerSec	Количество ошибок при проверке сообщения из-за неправильной контрольной суммы CRC
	InvalidOffsetOrSequenceRecordsPerSec	Количество ошибок при проверке сообщения из-за непостоянного смещения или порядкового номера в пакете
	MessagesInPerSec	Суммарное количество входящих сообщений
	NoKeyCompactedTopicRecordsPerSec	Количество ошибок при проверке сообщения из-за того, что для сжатого топика не указан ключ в секунду
	ProduceMessageConversionsPerSec	Количество конверсий сообщений в секунду для брокера
	ReassignmentBytesInPerSec	Количество байтов входящего трафика переназначения в секунду
	ReassignmentBytesOutPerSec	Количество байтов исходящего трафика переназначения в секунду
	ReplicationBytesInPerSec	Количество байтов передачи данных от других брокеров в секунду
	ReplicationBytesOutPerSec	Количество байтов, переданных для других брокеров в секунду
	TotalFetchRequestsPerSec	Количество запросов на получение в секунду
	TotalProduceRequestsPerSec	Количество запросов на производство сообщений в секунду
controller	ActiveControllerCount	Количество активных контроллеров
	ControllerState	Статус контроллера
	GlobalPartitionCount	Общее количество партиций
	GlobalTopicCount	Общее количество топиков
	OfflinePartitionsCount	Количество партиций, которые не имеют активного лидера и, следовательно, недоступны для записи или чтения
	PreferredReplicaImbalanceCount	Количество случаев дисбаланса в предпочтительной реплике
	ReplicasIneligibleToDeleteCount	Количество реплик, не соответствующих требованиям и ожидающих удаления
	ReplicasToDeleteCount	Количество реплик, ожидающих удаления
	TopicsIneligibleToDeleteCount	Количество топиков, не соответствующих требованиям и ожидающих удаления
	TopicsToDeleteCount	Количество топиков, ожидающих удаления
controllerStats Метрики группы имеют следующие атрибуты: • количество событий; • скорость за 15 мин; • скорость за 5 мин; • скорость за 1 мин; • средняя скорость; • 50/75/95/98/999/99-ый процентили выборки; • максимальное значение; • среднее значение; • минимальное значение; • StdDev.	AutoLeaderBalanceRateAndTimeMs	Отслеживает частоту и продолжительность автоматической перебалансировки лидера
	ControlledShutdownRateAndTimeMs	Отслеживает частоту и продолжительность контролируемых остановок
	ControllerChangeRateAndTimeMs	Отслеживает частоту и продолжительность изменений контроллера
	ControllerShutdownRateAndTimeMs	Отслеживает частоту и продолжительность остановки контроллера
	IsrChangeRateAndTimeMs	Отслеживает частоту и продолжительность задержки контроллера для ручной балансировки лидеров
	LeaderAndIsrResponseReceivedRateAndTimeMs	Отслеживает частоту и продолжительность получения ответа лидера и синхронизированных реплик (In-Sync Replicas, ISR)
	LeaderElectionRateAndTimeMs	Отслеживает частоту и продолжительность избрания лидера
	ListPartitionReassignmentRateAndTimeMs	Список скорости и времени переназначения партиций
	LogDirChangeRateAndTimeMs	Отслеживает частоту и продолжительность изменения LogDir (CPU latency)
	ManualLeaderBalanceRateAndTimeMs	Отслеживает частоту и продолжительность ручной балансировки лидеров (CPU latency)
	PartitionReassignmentRateAndTimeMs	Отслеживает частоту и продолжительность переназначения партиций (CPU latency)
	TopicChangeRateAndTimeMs	Отслеживает частоту и продолжительность создания новых топиков (CPU latency)
	TopicDeletionRateAndTimeMs	Отслеживает частоту и продолжительность удаления топиков (CPU latency)
	TopicUncleanLeaderElectionEnableRateAndTimeMs	Отслеживает частоту и продолжительность выбора **unclean-лидера** топика (лидера, назначенного после отключения предыдущего лидера, не имеющего реплик)
	UncleanLeaderElectionEnableRateAndTimeMs	Отслеживает частоту и продолжительность выбора unclean-лидера (CPU latency)
	UncleanLeaderElectionsPerSec	Количество выборов unclean-лидера в секунду
	UpdateFeaturesRateAndTimeMs	Отслеживает частоту и продолжительность обновления функции (CPU latency)
memory	HeapMemoryUsage_committed	Использование динамической памяти (кучи) — назначенная к использованию (в МБ)
	HeapMemoryUsage_init	Использование динамической памяти (кучи) — инициализация (в МБ)
	HeapMemoryUsage_max	Использование динамической памяти (кучи) — максимум (в МБ)
	HeapMemoryUsage_used	Использованная динамическая (куча) память (в МБ)
	NonHeapMemoryUsage_committed	Использование non-heap памяти — назначенная к использованию (в МБ)
	NonHeapMemoryUsage_init	Использование non-heap памяти — инициализация (в МБ)
	NonHeapMemoryUsage_max	Использование non-heap памяти — максимум (в МБ)
	NonHeapMemoryUsage_used	Использованная нединамическая память (куча) (в МБ)
	ObjectPendingFinalizationCount	Количество объектов, для которых ожидается доработка
network	NetworkProcessorAvgIdlePercent	Средняя доля времени, в течение которого потоки сетевого процессора простаивают. Значения находятся в диапазоне от `0` (все ресурсы используются) до `1` (все ресурсы доступны)
	RequestQueueSize	Размер очереди запросов
	ResponseQueueSize	Размер очереди ответов
session Метрики группы имеют следующие атрибуты: • количество событий; • скорость за 15 мин; • скорость за 5 мин; • средняя скорость; • скорость за 1 мин.	ZooKeeperAuthFailuresPerSec	Показывает количество неудачных (из-за ошибок авторизации) попыток подключиться к кластеру в секунду
	ZooKeeperDisconnectsPerSec	Показывает количество отключений клиента ZooKeeper от кластера в секунду. Данная метрика сообщает, отключается ли брокер, но не сообщает, работает ли ZooKeeper
	ZooKeeperExpiresPerSec	Показывает количество истекших сроков действия сеансов ZooKeeper в секунду. Наиболее точно показывает работоспособность системы
	ZooKeeperReadOnlyConnectsPerSec	Показывает количество подключений клиента к серверу, который является "Read Only" (не является ни "follower", ни "leader") в секунду. Клиент может только читать состояние ZooKeeper
	ZooKeeperSaslAuthenticationsPerSec	Показывает количество успешных прохождений аутентификации клиента в секунду
	ZooKeeperSyncConnectsPerSec	Показывает количество успешных подключений клиента ZooKeeper к кластеру в секунду

MiNiFi

Группа метрик	Название метрики	Описание
MinifiAgentCollector/Instance	activeThreads	Количество активных потоков
	bytesRead	Количество прочитанных байтов
	bytesSent	Количество отправленных байтов
	queuedContentSize	Размер контента в очереди
	queuedCount	Количество байтов в очереди
MinifiAgentCollector/SystemDiagnostics	availableProcessors	Доступные процессоры
	diskUtilization	Использование диска (в %)
	heapUtilization	Использование динамической памяти (кучи) (в %)
	loadAverage	Средняя нагрузка (в %)
	usedHeap	Использованная куча памяти (в МБ)
	usedSpaceGeneral	Использование дискового пространства (в МБ)

NiFi

Группа метрик	Название метрики	Описание
heap	committed	Использование динамической памяти (кучи) — назначенная к использованию (в МБ)
	init	Использование динамической памяти (кучи) — инициализация (в МБ)
	max	Использование динамической памяти (кучи) — максимум (в МБ)
	usage	Использование динамической памяти (кучи) (в МБ)
	used	Использованная динамическая память (куча) (в МБ)
non-heap	committed	Использование non-heap памяти — назначенная к использованию (в МБ)
	init	Использование non-heap памяти — инициализация (в МБ)
	max	Использование non-heap памяти — максимум (в МБ)
	usage	Использование non-heap памяти (в МБ)
	used	Использованная non-heap память (в МБ)
pools → Code-Cache	committed	Область памяти Code-Cache — назначенная к использованию (в МБ)
	init	Область памяти Code-Cache — инициализация (в МБ)
	max	Область памяти Code-Cache — максимум (в МБ)
	usage	Использование области памяти Code-Cache (в МБ)
	used	Использованная область памяти Code-Cache (в МБ)
pools → Compressed-Class-Space	committed	Область памяти Compressed-Class-Space — назначенная к использованию (в МБ)
	init	Область памяти Compressed-Class-Space — инициализация (в МБ)
	max	Область памяти Compressed-Class-Space — максимум (в МБ)
	usage	Использование области памяти Compressed-Class-Space (в МБ)
	used	Использованная область памяти Compressed-Class-Space (в МБ)
pools → G1-Eden-Space	committed	Область памяти G1-Eden-Space — назначенная к использованию (в МБ)
	init	Область памяти G1-Eden-Space — инициализация (init) (в МБ)
	max	Область памяти G1-Eden-Space — максимум (в МБ)
	usage	Использование (usage) области памяти G1-Eden-Space (в МБ)
	used	Использованная область памяти G1-Eden-Space (в МБ)
	used-after-gc	Использованная после запуска GC область памяти G1-Eden-Space (в МБ)
pools → G1-Old-Gen	committed	Область памяти G1-Old-Gen — назначенная к использованию (в МБ)
	init	Область памяти G1-Old-Gen — инициализация (init) (в МБ)
	max	Область памяти G1-Old-Gen — максимум (в МБ)
	usage	Использование (usage) области памяти G1-Old-Gen (в МБ)
	used	Использованная область памяти G1-Old-Gen (в МБ)
	used-after-gc	Использованная после запуска GC область памяти G1-Old-Gen (в МБ)
pools → G1-Survivor-Space	committed	Область памяти G1-Survivor-Space — назначенная к использованию (в МБ)
	init	Область памяти G1-Survivor-Space — инициализация (init) (в МБ)
	max	Область памяти G1-Survivor-Space — максимум (в МБ)
	usage	Использование (usage) области памяти G1-Survivor-Space (в МБ)
	used	Использованная область памяти G1-Survivor-Space (в МБ)
	used-after-gc	Использованная после запуска GC область памяти G1-Survivor-Space (в МБ)
pools → Metaspace	committed	Область памяти Metaspace — назначенная к использованию (в МБ)
	init	Область памяти Metaspace — инициализация (init) (в МБ)
	max	Область памяти Metaspace — максимум (в МБ)
	usage	Использование области памяти Metaspace (в МБ)
	used	Использованная область памяти Metaspace (в МБ)
total	committed	Назначенная к использованию общая память (в МБ)
	init	Инициализация общей памяти (в МБ)
	max	Максимум общей памяти (в МБ)
	usage	Использование общей памяти (в МБ)
	used	Использованная общая память (в МБ)
—	ActiveThreads	Количество активных потоков
—	BytesQueued	Количество байтов в очереди
—	BytesReadLast5Minutes	Количество байтов, прочитанных за последние 5 минут
—	BytesReceivedLast5Minutes	Количество байтов, полученных за последние 5 минут
—	BytesSentLast5Minutes	Количество байтов, отправленных за последние 5 минут
—	BytesWrittenLast5Minutes	Количество байтов, записанных за последние 5 минут
—	FlowFilesQueued	Количество FlowFiles в очереди
—	FlowFilesReceivedLast5Minutes	Количество FlowFiles, полученных за последние 5 минут
—	FlowFilesSentLast5Minutes	Количество FlowFiles, отправленных за последние 5 минут
—	TotalTaskDurationNanoSeconds	Общая продолжительность задачи (в наносекундах)

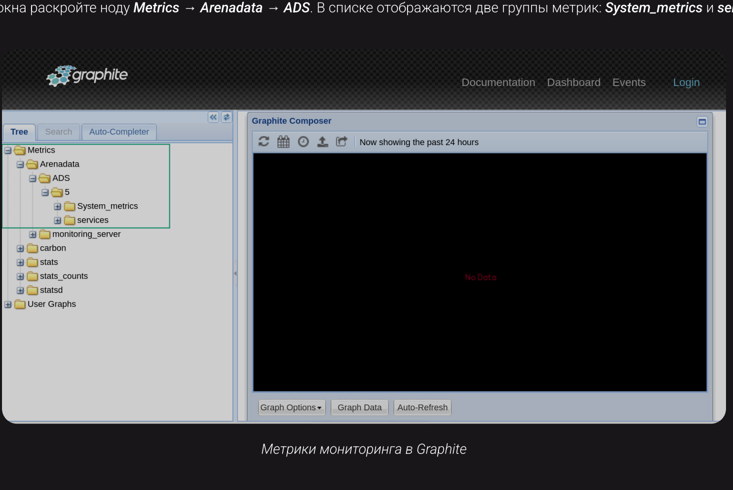
Просмотр метрик мониторинга

Вы можете использовать веб-интерфейсы Graphite и Grafana для просмотра метрик мониторинга.

Graphite

Graphite хранит числовые данные временных рядов и отображает графики данных по запросу. Чтобы просмотреть метрики, введите адрес хоста с кластером мониторинга в адресную строку браузера. Пример: *http://10.92.16.223*. Если вы не указали другой порт во время настройки сервисов мониторинга, Graphite использует порт по умолчанию (80). Если вы задали ***Web-interface TCP port*** при настройке Graphite, укажите номер порта через двоеточие. Пример: *http://10.92.16.223:8080*.

В левой части открывшегося окна раскройте ноду ***Metrics → Arenadata → ADS***. В списке отображаются две группы метрик: ***System_metrics*** и ***services***.

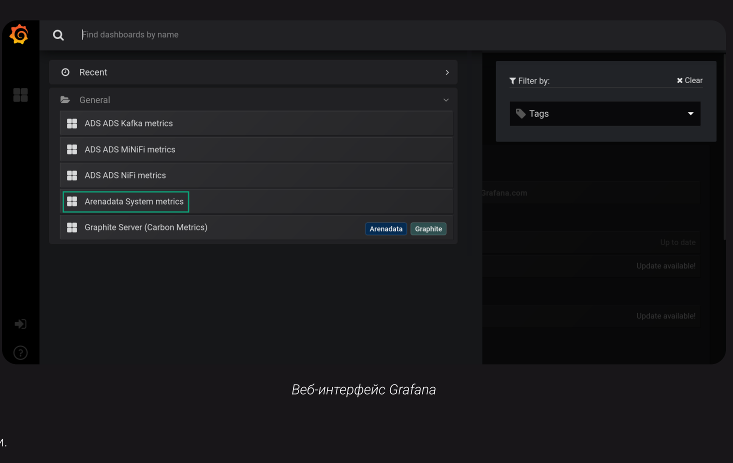


Метрики мониторинга в Graphite

Grafana

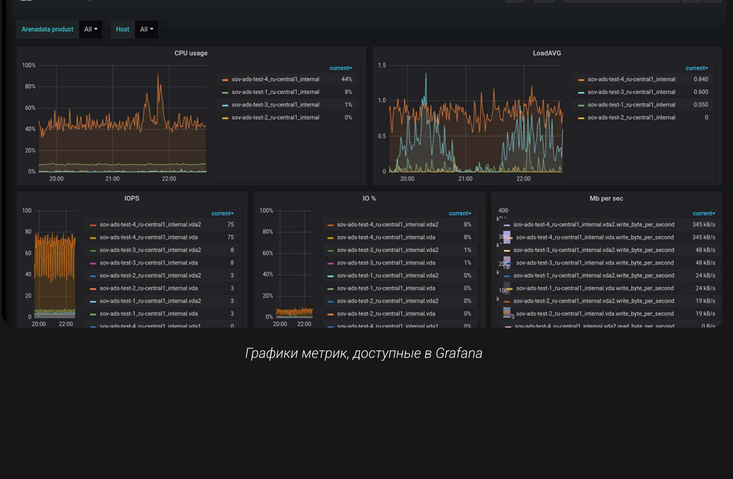
Grafana позволяет выполнять запросы данных и визуализировать метрики, хранящиеся в Graphite. Чтобы открыть веб-интерфейс Grafana, введите адрес хоста кластера мониторинга в строку браузера. Если вы не указали порт во время настройки сервисов мониторинга, Grafana использует порт 3000. Пример: *http://10.92.16.223:3000*. Если вы задали ***Port*** при настройке Grafana, укажите номер порта через двоеточие. Пример: *http://10.92.16.223:8081*.

В открывшейся форме нажмите на пункт меню ***Home***. Выберите один из дашбордов мониторинга, чтобы просмотреть метрики.



Веб-интерфейс Grafana

Откроется форма с графиками.



Графики метрик, доступные в Grafana

Клиенты для работы с Kafka

Ольга Семяне

Назад к содержанию

Клиенты (clients) для работы с Apache Kafka доступны на многих языках программирования.

В ADS предусмотрена возможность установки клиентских библиотек для различных языков, которые обеспечивают как низкоуровневый доступ к Kafka, так и потоковую обработку более высокого уровня.

Клиенты Kafka позволяют создавать распределенные приложения и микросервисы, которые считывают, записывают и обрабатывают потоки событий параллельно, в масштабе и отказоустойчивым образом даже в случае проблем с сетью или сбоев машины.

Kafka поставляется с некоторыми такими клиентами, которые дополняются десятками клиентов, предоставляемых сообществом Kafka.

ПРИМЕЧАНИЕ

Информация о клиентах Kafka приведена в качестве справочной информации. Представленные клиенты являются независимыми проектами с открытым исходным кодом.

Ниже приведен перечень известных и поддерживаемых на данный момент клиентов Kafka.

Язык реализации	Клиент	Описание
C/C++	librdkafka	Надежная высокопроизводительная библиотека C/C++ с полной поддержкой протоколов, поверх которой построено множество других языковых привязок, включая Haskell, Node.js, OCaml, PHP, Python, Ruby, C#/.NET
	libkafka	Собственная библиотека C++ с поддержкой протоколов для метаданных, производства, выборки и смещения
	libkafka-asio	Клиентская библиотека Kafka только для заголовков C++ с использованием Boost Asio
	csi-kafka	Асинхронная библиотека producer/consumer C++ 11 для Apache Kafka на основе Boost Asio
	libasynckafkaclient	Однопоточная асинхронная библиотека на основе C++
	kafka-cpp	Клиент C++ для Kafka. Библиотека, позволяющая создавать сообщения для распределенной службы обмена сообщениями публикации/подписки Kafka
Python	confluent-kafka-python Docs	Высокопроизводительный клиент Python на основе librdkafka с полной поддержкой протокола
	kafka-python	Реализация на чистом Python с полной поддержкой протокола. Включены реализации consumer и producer, поддерживается сжатие GZIP, LZ4 и Snappy
	pykafka	Драйвер Python с полной поддержкой протокола, Pythonic API и управляемой сбалансированной реализацией consumer
	pykafka	Позволяет создавать сообщения для распределенной службы обмена сообщениями публикации/подписки Kafka
	pykafka	Удобный для программиста клиент Kafka для Python. Он включает реализации producer и consumer Kafka на Python, которые дополнительно поддерживаются расширением, созданным на основе librdkafka
	pykafkap	Клиент Python от Urban Airship — простой клиент producer Kafka для Python
	brod	Позволяет создавать сообщения для распределенной службы обмена сообщениями публикации/подписки Kafka (producer, simple consumer, ZK-consumer)
Go (AKA golang)	sarama	Клиентская библиотека Go для Apache Kafka, лицензированная MIT.Чистая реализация с полной поддержкой протокола. Включены реализации consumer и producer, поддерживается сжатие GZIP и Snappy
	confluent-kafka-go Docs	Клиент Kafka Confluent для Golang включает в себя библиотеку librdkafka, обеспечивая полную поддержку протокола Kafka с высокой производительностью и надежностью
	franz-go Docs	Реализация Pure Go обеспечивает полную поддержку протокола, высокую производительность и поддержку всех KIP, обращенных к клиенту
Erlang	brod	Клиентская библиотека Kafka на Erlang. Полная поддержка потребительского протокола, очень эффективная реализация producer
	ekaf	Минималистичный высокопроизводительный клиент Kafka на Erlang
	erlkafka kafka-erlang	Клиент Kafka, написанный на Erlang
PowerShell	KNet Docs	Клиент PowerShell с поддержкой Producer, Consumer, Admin. Дополнительно из PowerShell можно запускать все доступные инструменты. Полный список доступных команд по использованию KNet PowerShell можно найти в статье KNet: PowerShell Module
.NET	KNet	Комплексный пакет .NET для Apache Kafka, предоставляющий все функции: Producer, Consumer, Admin, Streams, Connect, серверные части (ZooKeeper и Kafka)
	confluent-kafka-dotnet	Полнофункциональный клиент .NET для Apache Kafka на основе librdkafka
	kafka-net	Клиент C# с полной поддержкой протокола. Включает в себя consumer, producer, компоненты более низкого уровня и поддержку gzip (без быстрой загрузки)
	kafka-net	Реализация .NET клиента для Kafka с использованием C# для Kafka 0.8, охватывает большинство основных функций, включая простые producer и consumer
	CSharpClient-for-Kafka	Реализация .NET клиента для Kafka, обеспечивающая базовую функциональность через классы Producer/Consumer. Проект также предлагает сбалансированную потребительскую реализацию. Проект является ответвлением клиента ExactTarget Kafka-net
	kafka4net	Асинхронный клиент C#, поддерживающий сжатие и отслеживание изменения партиции-лидера
	kafka-sharp	Высокопроизводительный драйвер .NET Kafka
	kafunk	Полнофункциональный собственный клиент F#, работающий на .NET Core
Clojure	kafka-fast	Быстрый Kafka API для языков JVM, реализованный в Clojure
	clj-kafka	Оболочка для Java API, обеспечивающая взаимодействие с Kafka
	kinsky Docs	Клиентская библиотека Kafka clojure
	clj-kafka kafka-clj	Предоставляет producer и consumer, которые поддерживают базовый API выборки, а также интерфейс управляемой последовательности
Ruby	Karafka	Многопоточная эффективная среда обработки Kafka на Ruby и Rails. На основе librdkafka
	WaterDrop	Автономная библиотека, часть Karafka для создания сообщений Kafka
	Racecar	Простой фреймворк для написания consumer Kafka на Ruby, который хорошо интегрируется с Rails, на основе ruby-kafka
	DeliveryBoy	Простой способ публиковать сообщения в Kafka из приложений Ruby, на основе ruby-kafka
	EM-Kafka	Драйвер EventMachine для Kafka
	ruby-kafka	Клиентская библиотека Ruby для Apache Kafka
Node.js	node-rdkafka	Высокопроизводительный клиент NodeJS для Apache Kafka, являющийся оболочкой собственной библиотеки librdkafka
	Kafka-node	Клиент NodeJS с интеграцией ZooKeeper
	node-kafka 0.8-plus	Реализация JavaScript для NodeJS Server с поддержкой Vagrant и Docker
	kafka-java-bridge Docs	Оболочка Nodejs для JAVA Kafka API
	Prozess Docs	Поддержка низкоуровневого протокола в Node.js
	franz-kafka	Альтернативный node-клиент от Tagged
Proxy (HTTP REST, etc)	Dory	Daemon-producer для Apache Kafka, упрощает работу клиентов, отправляющих сообщения в Kafka, освобождая их от сложности прямого взаимодействия с кластером Kafka
	dropwizard-kafka-http	Этот проект создает конечную точку HTTP Apache Kafka для создания и использования сообщений. Он построен на платформе Dropwizard, которая делает службы Java REST удобными для оператора и простыми в развертывании
	Kafka HTTP	Конечная точка Rest предоставляет доступ к собственным высокоуровневым API-интерфейсам consumer и producer Scala
	Kafka REST Proxy Docs	Предоставляет интерфейс RESTful для кластера Kafka
	Kafka-Pixy	Proxy-сервер с двойным API (gRPC и REST) для Kafka с автоматическим контролем consumer groups. Он разработан, чтобы скрыть сложность клиентского протокола Kafka и предоставить простой API, который тривиально реализуется на любом языке
	Kastle	Эффективный Proxy-сервер Kafka REST для producer
	KastleX	Полнофункциональный Proxy-сервер HTTP REST для Kafka в Elixir с аутентификацией на основе JWT
Perl	Kafka Docs	Включены реализации Pure Perl, producer и consumer, интеграция с ZooKeeper, поддерживается сжатие GZIP и Snappy
stdin/stdout	kcat	Универсальный producer и consumer без JVM для Apache Kafka
PHP	phpkafka	Расширение PHP, построенное на librdkafka
	PHP Kafka client	Стабильный, готовый к работе и быстрый клиент Kafka для PHP на основе librdkafka
	log4php-kafka Kafka PHP	Библиотека PHP с consumer (простой и на основе ZooKeeper), producer и поддержкой сжатия
Rust	Kafka Rust Client Docs	Реализация Pure Rust с поддержкой запросов Metadata, Produce, Fetch и Offset. Поддерживает сжатие Gzip и Snappy
Alternative Java	Krackle	Клиент Kafka, предназначенный для минимизации количества создаваемых объектов и, следовательно, для сокращения накладных расходов на сборку "мусора"
Storm	storm-kafka-0.8-plus	Клиент Kafka, является частью Apache Storm
Scala DSL	scala-kafka	DSL для разработчиков Scala, который можно использовать для создания и использования сообщений с помощью Kafka
Swift	Franz	Клиент Kafka для iOS и macOS, реализующий swift producer, а также simple и high-level consumer

Kafka API

Ольга Семме

Содержание

Назад к содержанию

Обзор
Установка зависимостей
Producer API
Consumer API
Streams API
Connect API
Admin API

Обзор

Kafka включает в себя пять основных API:

- Producer API
- Consumer API
- Streams API
- Connect API
- Admin API

API в Kafka используются для реализации клиентских приложений, которые потребляют и передают данные в Kafka, управляют брокерами и топиками Kafka, а также выполняют потоковую обработку.

Клиенты для приложений Java поддерживаются как часть основного проекта Kafka, остальные доступны как независимые проекты с открытым исходным кодом.

ПРИМЕЧАНИЕ

Список клиентов, отличных от Java, приведен в статье Клиенты для работы с Kafka.

Установка зависимостей

Для создания API в Kafka используется Apache Maven — инструмент, основанный на концепции объектной модели проекта POM (Project Object Model), управляющий сборкой проекта, загружающий подходящие версии пакетов и библиотеки. Для загрузки необходимого пакета необходимо установить зависимость (dependency) в файле pom.xml. На основе заданной зависимости Maven загрузит и добавит JAR-файл (Java-архив) к пути Java. После этого Java сможет найти и использовать классы, заданные в файле JAR.

Для указания зависимости используются параметры GAV (groupId, artifactId, version):

- `groupId` — идентификатор производителя объекта. Часто используется схема, принятая в обозначении пакетов Java.
- `artifactId` — идентификатор объекта. Обычно это имя создаваемого модуля или приложения.
- `version` — версия описываемого объекта.

Для использования API зависимости Maven в файле *pom.xml* устанавливаются следующим образом:

```
<dependency>
        <groupId>org.apache.kafka</groupId>
        <artifactId>kafka-clients</artifactId>
        <version>3.5.0</version>
</dependency>
```

Producer API

Producer API позволяет приложениям отправлять потоки данных в топики в кластере Kafka.

Consumer API

Consumer API позволяет приложениям считывать потоки данных из топиков в кластере Kafka.

Streams API

Streams API позволяет преобразовывать потоки данных из входных топиков в выходные топики.

ПРИМЕЧАНИЕ

- Для получения дополнительной информации о Kafka Streams можно обратиться к статье KAFKA STREAMS.
- Для получения информации о создании приложения Kafka Streams с использованием кластера ADS можно обратиться к статье Создание приложения Kafka Streams.

Connect API

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. Предварительно созданные коннекторы используются без необходимости написания кода.

ПРИМЕЧАНИЕ

- Для получения дополнительной информации о Kafka Connect можно обратиться к статье KAFKA CONNECT.
- Для получения информации о работе с Kafka Connect при помощи инструментов интерфейса ADS Control можно обратиться к статьям:
  - Обзор UI ADS Control
  - Mirror Maker 2 в ADS Control

Admin API

Admin API поддерживает управление топиками, брокерами, списками управления доступом и другими объектами Kafka, а также их проверку.

ПРИМЕЧАНИЕ

Полный список доступных API приведен в статье kafka 2.8.1 API

NiFi REST API

Ольга Семме

Назад к содержанию

REST API обеспечивает программный доступ для управления экземпляром NiFi в режиме реального времени.

REST API позволяет при помощи REST-запросов без использования пользовательского интерфеса NiFi выполнять различные операции, например:

- работать с атрибутами и параметрами потока данных;
- производить мониторинг потока и элементов потока данных;
- выполнять настройку элементов потока.

Ниже приведено описание групп API, представленных в NiFi.

Группа API	Назначение
Access User	Конечные точки аутентификации и токенов
Connections	Управление и получение конфигурации соединений, например: - установка приоритета очереди; - обновление назначения подключения.
Controller	Управление конфигурацией контроллера, например: - вывод текущей конфигурации; - управление кластером; - создание задач отчетности.
Controller Services	Управление сервисами контроллера, обновление ссылок на сервисы контроллера
Counters	Управление счетчиками событий: - получение счетчиков событий; - сброс счетчиков событий.
Data Transfer	Управление транзакциями, например: - управление сроком жизни транзакции; - отмена указанной транзакции; - отправка данных; - получение данных.
Flow	Управление конфигурацией потока, например: - получение данных о потоке (например, список созданных бюллетеней); - получение данных о статусе компонентов потока; - получение истории запросов.
FlowFile Queues	Управление и получение конфигураций очереди потока данных, например: - просмотр содержимого очереди; - загрузка содержимого потокового файла; - очищение очереди.
Funnels	Управление "воронками"
Input Ports	Управление и получение конфигураций входных портов, например: - создание входного порта; - установка контроля удаленного доступа к порту.
Labels	Управление метками, например: - создание метки; - установка стиля метки.
Output Ports	Управление и получение конфигураций выходных портов, например: - создание выходного порта; - установка контроля удаленного доступа к порту.
Parameter Contexts	Создание, удаление, управление и получение конфигураций контекстов параметров
Parameter Providers	Управление и получение конфигураций поставщиков параметров
Process Groups	Управление и получение конфигураций группы процессов, например: - создание компонентов; - создание экземпляра шаблона; - загрузка шаблона.
Processors	Управление и получение конфигураций процессоров, например: - создание и удаление процессоров; - установка свойств процессора, в том числе установка расписания работы процессора.
Policies	Управление и получение данных о политиках доступа
Provenance	Управление происхождением данных (provenance): - запрос о происхождении; - поиск происхождения данных.
Provenance Events	Управление происхождением событий: - загрузка содержимого события; - повтор запуска содержимого из события заданного происхождения.
Remote Process Groups	Управление и получение конфигураций удаленных групп, например: - создание удаленной группы; - включение передачи данных.
Reporting Tasks	Управление задачами отчетности
Resources	Получение доступных ресурсов, поддерживающих политики доступа/авторизации
Site to Site	Получение данных для связи "site to site": - получение сведений об экземпляре NiFi, необходимых для связи (доступные порты); - получение сведений о доступных участниках одноранговой сети и статусе их NiFi-экземпляра.
Snippets	Управление фрагментами схемы: - перемещение фрагмента; - удаление фрагмента.
System Diagnostics	Получение диагностики системы
Templates	Управление шаблонами, например: - загрузка шаблона; - удаление шаблона.
Tenants	Управление группами пользователей (арендаторами): - добавление пользователей; - создание групп пользователей.
Versions	Управление версиями групп процессов

ПРИМЕЧАНИЕ

Полный список Rest API, доступных в NiFi, с указанием запросов и конечных точек приведен в статье /nifi-api.

Аппаратные требования к серверам

Ольга Сычева

Содержание

Назад к содержанию

ZooKeeper, Kafka Manager, Schema Registry
Kafka Broker
NiFi
ksqlDB
REST Proxy
Arenadata Enterprise Tools
Arenadata Monitoring

ВАЖНО

Системные требования, приведенные ниже, являются **минимальными**. Целевой сайзинг необходимо рассчитывать исходя из требований вашей организации.

Приоритетнее всего использовать физические серверы, но необходимо проанализировать энергопотребление предполагаемого оборудования.

Arenadata Streaming одинаково эффективно работает как на bare metal, так и в облаке.

При обеспечении сетевой доступности между серверами возможны также следующие сценарии:

- гетерогенная ИТ-инфраструктура,
- Multi-clouds,
- распределенная установка на разные инфраструктуры.

Последовательность подготовки физических и виртуальных хостов приведена в статье Подготовка хостов.

Примеры требований к разным сервисам в зависимости от профиля нагрузки показаны ниже.

ZooKeeper, Kafka Manager, Schema Registry

Minimal

Хранение	CPU	RAM	Сеть
400 ГБ SSD для ОС 500 ГБ SSD для данных	8 ядер	32 ГБ	1 x 10 GbE

Proof of concept

Количество узлов	Хранение	CPU	RAM	Сеть
3 и более	1 x 500 ГБ HDD для ОС 1 x 2.4 ТБ SSD для журналов транзакций 1 x 1 ТБ HDD для данных	8 ядер	32 ГБ	1 x 10 GbE

Commodity

Количество узлов	Хранение	CPU	RAM	Сеть
3 и более	2 x 500 ГБ HDD RAID1 для ОС 2 x 2.4 ТБ SSD RAID1 для журналов транзакций 2 x 1 ТБ HDD RAID1 для данных	8 ядер	32 ГБ	2 x 10 GbE

Balanced

Количество узлов	Хранение	CPU	RAM	Сеть
3 и более	2 x 500 ГБ HDD RAID1 для ОС 2 x 2.4 ТБ SSD RAID1 для журналов транзакций 2 x 1 ТБ HDD RAID1 для данных	8 ядер	32 ГБ	2 x 10 GbE

Compute-intensive

Количество узлов	Хранение	CPU	RAM	Сеть
3 и более	2 x 500 ГБ HDD RAID1 для ОС 2 x 2.4 ТБ SSD RAID1 для журналов транзакций 2 x 1 ТБ HDD RAID1 для данных	8 ядер	32 ГБ	2 x 10 GbE

ВАЖНО

Для обеспечения HA (High Availability) необходимо, чтобы количество экземпляров сервиса ZooKeeper было нечетным.

Kafka Broker

Proof of concept

Количество узлов	Хранение	CPU	RAM	Сеть
3 и более	1 x 500 ГБ HDD для ОС 2 x 500 ГБ SSD RAID1 для данных	8 ядер	32 ГБ	2 x 10 GbE

Commodity

Количество узлов	Хранение	CPU	RAM	Сеть
3 и более	2 x 500 ГБ HDD RAID1 для ОС 4 x 500 ГБ SAS12G SSD RAID10 (PCIe Gen3 x4+) для данных	8 ядер	32 ГБ	2 x 25 GbE

Balanced

Количество узлов	Хранение	CPU	RAM	Сеть
3 и более	2 x 500 ГБ HDD RAID1 для ОС 8 x 500 ГБ SAS12G SSD RAID10 (PCIe Gen3 x8) для данных	16 ядер	64 ГБ	2 x 25 GbE

Compute-intensive

Количество узлов	Хранение	CPU	RAM	Сеть
6 и более	2 x 500 ГБ HDD RAID1 для ОС 8 x 500 ГБ SAS12G SSD RAID10 (PCIe Gen3 x8) для данных	24 ядра	96 ГБ	2 x 25 GbE

NiFi

Minimal

Хранение	CPU	RAM	Сеть
400 ГБ SSD для ОС 500 ГБ SSD для данных	8 ядер	32 ГБ	1 x 10 GbE

Proof of concept

Количество узлов	Хранение	CPU	RAM	Сеть
2 и более	1 x 500 ГБ HDD для ОС 1 x 500 ГБ SSD для данных	8 ядер	32 ГБ	1 x 10 GbE

Commodity

Количество узлов	Хранение	CPU	RAM	Сеть
2 и более	2 x 500 ГБ SSD RAID1 для ОС 4 x 500 ГБ SAS12G SSD RAID10 (PCIe Gen3 x4+) для данных	12 ядер	64 ГБ	2 x 25 GbE

Balanced

Количество узлов	Хранение	CPU	RAM	Сеть
2 и более	2 x 500 ГБ SSD RAID1 для ОС 8 x 500 ГБ SAS12G SSD RAID10 (PCIe Gen3 x8) для данных	24 ядра	96 ГБ	2 x 25 GbE

Compute-intensive

Количество узлов	Хранение	CPU	RAM	Сеть
4 и более	2 x 500 ГБ SSD RAID1 для ОС 16 x 500 ГБ SAS12G SSD RAID10 (2 контроллера PCIe Gen3 x8) для данных	48 ядер	192 ГБ	2 x 25 GbE

ksqlDB

Профиль нагрузки	Количество узлов	Хранение	CPU	RAM
Все	2 и более	SSD (объем зависит от количества запросов и агрегаций)	4 ядра	32 ГБ

REST Proxy

Профиль нагрузки	Количество узлов	Хранение	CPU	RAM
Все	2 и более	HDD (без требования к скорости)	16 ядер и более (работа с producer/consumer)	1 ГБ 64 МБ на producer 16 МБ на consumer

Arenadata Enterprise Tools

Если вместе с вашим продуктом вы используете ET, примите во внимание следующие рекомендации:

- Виртуальный сервер.
- CPU: от 4 ядер.
- RAM: от 8 ГБ.
- Диск: от 50 ГБ.

Arenadata Monitoring

Если вместе с вашим продуктом вы используете ADM, примите во внимание следующие рекомендации:

- Виртуальный сервер.
- CPU: от 8 ядер.
- RAM: от 16 ГБ.
- Диск: от 10 ГБ на каждый отслеживаемый хост.

Аппаратные требования к сети

Ольга Семме

Назад к содержанию

1. Требуются одна или несколько выделенных одноранговых Interconnect-сетей для внутренней коммуникации кластера.
2. К Interconnect-сетям должны быть подключены все серверы кластера.
3. Все серверы должны иметь полное доменное имя (Fully Qualified Domain Name, FQDN) и возможность обмениваться данными друг с другом на заданных скоростях.
4. Должны быть известны IP-адреса каждого сервера.
5. Должна быть обеспечена доступность Ping любого из серверов (Internet Control Message Protocol, ICMP).
6. В случае инсталляции с доступом в Интернет требуется доступ к развернутому на выделенном сервере Arenadata Cluster Manager. Установка Arenadata Cluster Manager описана в Установка ADCM.
7. В случае инсталляции без доступа в Интернет (offline) требуется доступ к развернутому на выделенном сервере Arenadata Enterprise Tools. Установка Enterprise Tools описана в разделе Установка кластера Enterprise Tools.
8. На серверах, используемых для установки кластера, должны быть открыты порты, приведенные ниже.

ПРИМЕЧАНИЕ

Для ADS каждый порт относится к серверу, на который устанавливается конкретный сервис.

ADS

Сервис	Порт	Назначение
Общие	22	Порт для подключения к SSH-провайдеру
	2015	TCP-порт для отправки метрик на сервер мониторинга
	2016	UDP-порт для отправки метрик на сервер мониторинга
ZooKeeper	2181	Порт доступа к сервису
	2888 3888	Порты межсерверного взаимодействия для компонентов кворума
Kafka	9092	HTTP-порт доступа к сервису
	9093	HTTPS-порт доступа к сервису
	9999	JMX-порт для доступа к метрикам сервиса
Kafka REST Proxy	8082	Порт доступа к сервису
	9998	JMX-порт для доступа к метрикам сервиса
Schema-Registry	8081	Порт доступа к сервису
	9997	JMX-порт для доступа к метрикам сервиса
ksqlDB	8088	Порт доступа к компоненту ksqlDB Server
NiFi	9090	HTTP-порт доступа к UI сервиса
	9091	HTTPS-порт доступа к UI сервиса
	11443	Порт взаимодействия между серверами NiFi, объединенными в кластер
	18080	Порт доступа к UI компонента NiFi Registry
NiFi2	9090	HTTP-порт доступа к UI сервиса
	9091	HTTPS-порт доступа к UI сервиса
	11443	Порт взаимодействия между серверами NiFi2, объединенными в кластер
	18080	Порт доступа к UI компонента NiFi2 Registry

Monitoring

Порт	Назначение
3000	HTTP-порт, используемый Grafana. Значение по умолчанию может быть изменено при установке кластера мониторинга на странице конфигурации сервиса Grafana в разделе ***ip_and_ports***
80	TCP-порт, используемый Graphite. Значение по умолчанию может быть изменено при установке кластера мониторинга на странице конфигурации сервиса Graphite в разделе ***ip_and_ports***

Enterprise Tools

Порт	Назначение
443	TCP-порт, используемый Arenadata Enterprise Tools (служба реестра Docker)
81	TCP-порт Arenadata Enterprise Tools (доступ к репозиториям в случае offline-установки)

ADCM

Порт	Назначение
8000	TCP-порт для отправки статусов компонентов кластера в ADCM

ADPS

Все используемые порты для кластера ADPS указаны в статье Требования к сети.

Программные требования

Ольга Семме

Содержание

Назад к содержанию

- Операционная система
- Предустановленное ПО
  - Установка Java из репозитория Arenadata
- Особенности установки ADS
- SELinux
- Firewall
- Браузеры, поддерживающие сервис NiFi

Операционная система

ВАЖНО

- Жизненный цикл Centos 7.9 завершился 30 июня 2024.
- Ubuntu 22.04 заменяется на Ubuntu 24.04 в релизах после 1 апреля 2026 года.
- Alt Linux 8.4 SP заменяется на Alt Linux 10 SP в релизах после 1 января 2026 года.

1. ADS поддерживает только архитектуру x86_64 и следующие операционные системы:
   - Centos 7.9;
   - RHEL 7.9;
   - Ubuntu 22.04;
   - Alt Linux 8.4 SP — доступно только в Enterprise-версии;
   - Alt Linux 10 SP — доступно только в Enterprise-версии;
   - Astra Linux 1.7 "Орел" SE с Axiom JDK — доступно только в Enterprise-версии;
   - Astra Linux 1.7 "Воронеж" SE с Axiom JDK — доступно только в Enterprise-версии;
   - РЕД ОС 7.3 Сертифицированная редакция — доступно только в Enterprise-версии.
2. Необходимо подготовить все зависимые репозитории дополнительно к выбранной операционной системе.
3. Со всех машин в кластере должен быть доступ к официальным репозиториям (или создано локальное зеркало):
   - CentOS Extras
   - CentOS Updates
   - CentOS Base
4. В случае online-установки необходимо обеспечить доступ к репозиториям, перечисленным на вкладке ***Configuration*** в меню кластера ADS в UI ADCM. Вкладка ***Configuration*** описана в статье Настройка кластера.
5. В случае offline-установки необходимо обеспечить доступ к хосту, на котором развернут кластер Enterprise Tools. Установка кластера Enterprise Tools описана в разделе Установка кластера Enterprise Tools.
6. Для поддержки kTLS (поддерживается начиная с ADS 3.9.0.1.b1) требуется ядро операционной системы версии 5.9 и выше. По умолчанию режим kTLS включен. Но если версия ядра ниже требуемой, то произойдет откат в обычный режим.
7. Для использования Astra Linux "Воронеж" сервисный пользователь ADCM должен иметь метку целостности `63`. `sudo pdpl-user -i 63 adcm`.

Предустановленное ПО

1. Должен быть предварительно настроен NTP на серверах.
2. Должен быть обеспечен доступ к пользователю `root` или любому пользователю с правами `sudo` на серверах кластера.
3. На хостах должна быть предустановлена Java 17 для всех сервисов и Java 21 для сервиса NiFi2.

В операционных системах Centos, RedHat и AstraLinux в системных пакетах отсутствует Java 17.

Для хостов с любой из таких ОС перед обновлением/установкой кластера воспользуйтесь одним из способов установки Java 17:

- Установите вручную Java 17 на хостах и настройте переменную JAVA_HOME.
- Установите Java из репозитория Arenadata способом, описанным ниже.

ВНИМАНИЕ

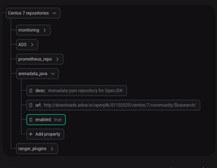
Перед установкой сервиса NiFi2 должна быть выполнена установка Java 21 из репозитория Arenadata **для всех операционных систем**. Ручной способ установки Java не подходит.

Установка Java из репозитория Arenadata

Начиная с версии ADS 3.7.2.1.b1 на странице настройки кластера доступна опция установки Java 17 и Java 21 из репозитория Arenadata.

Если опция активирована перед запуском действия ***Install***, ***Add/Remove components*** или ***Update*** для кластера или сервиса, последняя версия Java 17 и Java 21 будет установлена на хостах кластера во время выполнения действия.

Для активации опции установки Java 17 и Java 21 из репозитория Arenadata раскройте группу с репозиториями выбранной ОС (например, ***Centos 7 repositories***), выберите репозиторий ***arenadata_java*** и раскройте его параметры. Укажите значение `true` для параметра ***enabled***.



Настройка установки Java из репозитория Arenadata для Centos 7

После активации опции Java 17 и Java 21 будет установлена из репозитория Arenadata независимо от того, что указано в качестве переменной JAVA_HOME (группа ***Custom JDK settings***).

ВНИМАНИЕ

Arenadata не является официальным поставщиком JDK, и эта функция предоставляется для обеспечения полностью автоматической установки в системах, не имеющих необходимых зависимостей. В производственных средах мы рекомендуем использовать JDK от официальных поставщиков.

Особенности установки ADS

1. Процесс установки модифицирует файлы, которые не должны контролироваться configuration management system (при ее наличии):
   - системный файл */etc/hosts*;
   - системный файл */etc/selinux/config*;
   - системный файл */etc/sysctl.conf*.
2. Процесс установки создает файлы в */usr/lib/systemd/system/*, которые не должны контролироваться configuration management system (при ее наличии).
3. Сервисы, устанавливаемые в кластере, не должны контролироваться configuration management system (при ее наличии).
4. Рекомендуется отключать антивирусные программы перед установкой и в течение процесса эксплуатации ADS. Если антивирус все же используется — пожалуйста, уточните настройки, необходимые для использования ADS, у поставщика вашего антивирусного ПО.

SELinux

SELinux должен быть выключен. Полное руководство по этой операции вы можете найти в разделе Установка ADCM.

Firewall

Firewall должен быть остановлен. Полное руководство по этой операции вы можете найти в разделе Установка ADCM.

Браузеры, поддерживающие сервис NiFi

1. Работу сервиса NiFi поддерживают браузеры:
   - Chrome
   - FireFox
   - Edge
   - Safari
2. Пользовательский интерфейс поддерживается в текущей стабильной версии этих браузеров и в предыдущей. Например, если текущий стабильный выпуск — 45.X, то официально поддерживаемыми версиями будут 45.X и 44.X.
3. Для Safari, который выпускает major-обновления гораздо реже, поддерживаются только две последние версии.
4. Поддерживаемые версии браузеров зависят от возможностей пользовательского интерфейса и используемых им зависимостей. Функции пользовательского интерфейса разрабатываются и тестируются в поддерживаемых браузерах.

Содержание

Рекомендации по настройке брокеров Kafka
- Настройка репликации топиков
- Настройка партиций топиков
- Настройка политик хранения
- Настройка корректного завершения работы
- Настройка баланса лидерства

Рекомендации по работе с производителями и потребителями
- Настройка получения производителем подтверждения записи сообщения
- Проверка позиции потребителя
- Назначение топикам групп потребителей

Настройка безопасной среды

Аппаратное обеспечение
- Общие рекомендации при выборе оборудования
- Рекомендации при выборе памяти
- Рекомендации при выборе процессоров
- Хранение данных

Сеть

Программное обеспечение
- Общие рекомендации
- Файловые дескрипторы
- Виртуальная память

ПРИМЕЧАНИЕ

Для получения информации об основных принципах работы в Kafka можно обратиться к статье «Начало работы с Kafka».

Рекомендации по настройке брокеров Kafka

При установке ЛО в конфигурационном файле `/usr/local/kafka/config/server.properties` устанавливаются необходимые параметры для брокеров Kafka. Параметры доступны в комментариях к конфигурационному файлу Kafka, описание и статьи Kafka документации параметры для.

Некоторые параметры настройки, для их добавления следует использовать ключ **`--override`** и ключ группы **`server.properties`**, где требуется написать параметры и их значения. Например, для добавления параметра `num.partitions` со значением `3` необходимо использовать следующую команду:

```
/usr/local/kafka/bin/kafka-topics.sh --create --topic new-topic --bootstrap-server hostname:9092 --override num.partitions=3
```

Настройка репликации топиков

Репликация топиков определяется следующими параметрами:

- **`default.replication.factor`** — коэффициент репликации по умолчанию для автоматически создаваемых топиков.

Высокий коэффициент репликации увеличивает доступность данных. Однако, при коэффициенте репликации `3` и более Kafka реплицирует журнал для каждой партиции топика на трех брокерах, что может привести к уменьшению пропускной способности и увеличению задержки при записи данных.

Этот параметр может быть изменен в любое время.

Существует возможность установить репликацию для отдельного топика.

Пример установки коэффициента репликации для отдельного топика

Необходимо при создании топика использовать опцию `--replication-factor`.

```
$ /usr/lib/kafka/bin/kafka-topics.sh --create --topic new-topic --replication-factor 3 --bootstrap-server hostname:9092
```

При запуске информация для созданного топика, что топик имеет 3 реплики:

```
$ /usr/lib/kafka/bin/kafka-topics.sh --describe --topic new-topic --bootstrap-server hostname:9092

Topic: new-topic    TopicId: 0uHnOST3SUyvk4kNiXa_Pw PartitionCount: 1    ReplicationFactor: 3    Configs: unclean.
leader.election.enable=false
        Topic: new-topic    Partition: 0    Leader: 1002    Replicas: 1002,1001,1003    Isr: 1002,1001,1003
```

При необходимости возможно изменить коэффициент репликации у существующей партиции.

Пример изменения коэффициента репликации

Для изменения используем скрипт.

Содержимое JSON-файла:

```
$ sudo vim increase-replication-factor.json
```

Содержимое JSON-файла:

```
{"version":1,
 "partitions":[{"topic":"new-topic","partition":0,"replicas":[1002,1001,1003]}]}
```

где коэффициент репликации партиции `0` топика `new-topic` распространяется на брокеры `1002`, `1001`, `1003`.

Запустить скрипт, применяющий изменения конфигурации:

```
$ /usr/lib/kafka/bin/kafka-reassign-partitions.sh --zookeeper localhost:9092 --reassignment-json-file increase-
replication-factor.json --execute
```

- **`min.insync.replicas`** — минимальное количество синхронизированных реплик, которое должно находиться в синхронизации партиции топика, чтобы брокер принял сообщение.

Значение параметра `min.insync.replicas` должно быть меньше значения параметра **`default.replication.factor`**, иначе топики не будут создаваться. Если количество синхронизированных реплик для партиции станет меньше, чем указано в `min.insync.replicas`, то производитель получит исключение **`NotEnoughReplicas`** или **`NotEnoughReplicasAfterAppend`**.

Этот параметр может быть задан и конфигурации ADSM.

При совместном использовании **`min.insync.replicas`** и **`request.required.acks`** позволяет обеспечить более высокие гарантии надежности.

Успешный сценарий:

- топик с коэффициентом репликации `3`.
- **`min.insync.replicas`** — значением `2`.
- **`request.required.acks`** — значением **`all`** или **`-1`**.

Такая система гарантирует подтверждение записи, если все 3 реплики работают, или если выполнено хотя бы 2 из 3 реплик.

Пример установки параметров для отдельного топика

Необходимо при создании топика использовать опции **`--config`** с указанием параметра **`min.insync.replicas`**.

```
$ /usr/lib/kafka/bin/kafka-topics.sh --create --topic new-topic1 --config min.insync.replicas=3 --bootstrap-server
hostname:9092
```

Настройка партиций топиков

Параметр **`num.partitions`** определяет количество партиций по умолчанию для каждого создаваемого топика.

Любой топик может иметь только одну партицию, однако в большинстве параллельному потреблению. Для данных с каналами следует обратить изменение количества партиций в топике.

Этот параметр может быть изменен в любое время.

Существует возможность установить количество партиций для отдельного топика.

Пример установки количества партиций для отдельного топика

Необходимо при создании топика использовать опцию **`--partitions`**.

```
$ /usr/lib/kafka/bin/kafka-topics.sh --create --topic topic1 --partitions 3 --bootstrap-server hostname:9092
```

Настройка политик хранения

Для оптимизации хранения данных рекомендуется установить корректную политику хранения сообщений в топике.

Политику хранения определяют следующие параметры:

- **`log.cleanup.policy`** — определяет политику хранения для использования в сегментах журнала. Значение по умолчанию действует для всех пользовательских топиков. Может быть изменено в конфигурации ADSM по топику.

  Возможные значения:
  - **`delete`** — отбрасывает старые сегменты топика, когда он время хранения или предельный размер был достигнут.
  - **`compact`** — включает сжатие журнала, что сохраняет последнее значение для каждого ключа. Также можно указать обе политики в списке, разделенном запятыми (**`delete, compact`**). В этом случае старые сегменты будут сжиматься с учетом конфигурации времени хранения и размера журнала (**`retention.ms`** и **`retention.bytes`**), а новые сегменты будут сжиматься.

- **`log.retention.ms`** — определяет максимальное время, в течение которого сообщение в топике будет доступно, прежде чем будет удалены старые сегменты топика. Значение по умолчанию действует для всех пользовательских топиков. Может быть изменено в конфигурации ADSM по топику. Значение **`-1`** означает, что время хранения не ограничено. Параметр может быть задан и конфигурации ADSM.
- **`log.retention.bytes`** — определяет максимальный размер журнала, до которого должен достигнуть сегмент журнала, чтобы освободить место, если установлена политика хранения **`delete`**. По умолчанию нет ограничений по размеру, только по времени. Ограничение применяется к каждой партиции. Поэтому, если у вас есть топик с 8 партициями, и для этого параметра установлено значение 1 ГБ, максимальный объем данных, сохраненных для топика, составит 8 ГБ.

Существует возможность установить данные параметры для отдельного топика.

Пример установки политики хранения для отдельного топика

Необходимо при создании топика использовать опцию **`--config`** с указанием параметров **`cleanup.policy`**, **`retention.ms`**, **`retention.bytes`**.

```
$ /usr/lib/kafka/bin/kafka-topics.sh --create --topic test-topic1 --config cleanup.policy=delete --config retention.ms=10000 --
config retention.bytes=128000 --bootstrap-server localhost:9092
```

Настройка корректного завершения работы

Параметры **`controlled.shutdown.enable`** по умолчанию **`true`**, включает управление отключением серверов. Параметр может быть задан и интерфейса ADSM.

Параметр **`auto.leader.rebalance.enable`** по умолчанию **`true`**, позволяет автоматически распределять лидерство для партиций. Брокер периодически проверяет, есть ли брокеры в кластере, которые являются предпочтительными лидерами для любой из партиций, которые они хранят. Если это так, и эти брокеры-лидеры действительно синхронизированы, контроллер инициирует выборы, чтобы сделать их лидерами.

Параметр **`default.replication.factor`** может быть задан и значением `2` и более).

Настройка баланса лидерства

Всякий раз, когда брокер останавливается или аварийно завершает работу, распределение лидеров партиций нарушается. После перезапуска брокер становится ведомым для всех своих партиций. Это означает, что он не будет использоваться клиентами для чтения или записи, что нежелательно, и Kafka есть логика перезапуска для решения этой проблемы.

Для для этого параметра установить значение **`true`**, необходимо включить автоматическое распределение лидеров. Это можно сделать, установив параметр **`auto.leader.rebalance.enable`** на значение **`true`**.

Если для этого параметра установить значение **`false`**, необходимо вручную использовать скрипты в системе для распределения лидеров. Для этого необходимо выполнить команду:

```
$ /usr/lib/kafka/bin/kafka-leader-election.sh --bootstrap-server localhost:9092 --election-type preferred --all-topic-partitions
```

Рекомендации по работе с производителями и потребителями

Настройка получения производителем подтверждения записи сообщения

Параметр **`request.required.acks`** определяет, когда запрос на производство считается выполненным. К частности, сколько других брокеров должны подтвердить, что они видели сообщение, прежде чем оно будет подтверждено клиенту.

Возможные значения:

- **`0`** — производитель никогда не ждет подтверждения от брокера. Этот вариант обеспечивает наименьшую задержку, но гарантирует наименьшую надежность. Некоторые данные будут потеряны, если что-то сломается.
- **`1`** — производитель получит подтверждение после того, как ведущая реплика получит данные. Это значение по умолчанию.
- **`-1`** — производитель получит подтверждение после того, как все синхронизированные реплики получили данные. Этот вариант обеспечивает наилучшую устойчивость.

Совместное использование параметров **`min.insync.replicas`**, описанного выше, и **`request.required.acks`** позволяет обеспечить более высокие гарантии надежности.

Пример установки параметра

Необходимо при создании топика использовать опцию **`--request-required-acks`**.

```
$ kin/kafka-console-producer.sh --topic new-topic --request-required-acks -1 --bootstrap-server hostname:9092
```

Проверка позиции потребителя

При необходимости это можно сделать при помощи скрипта consumer-groups.sh, возможно увидеть позицию потребителей топика.

Пример получения списка потребителей топика

Команда для получения списка для группы потребителей

```
$ /usr/lib/kafka/bin/kafka-consumer-groups.sh --bootstrap-server localhost:9092 --describe --group new-group
```

В результате появился позиция потребителя в группе потребителей с именем **`new_group`**, а также то, насколько он отстает от конца журнала топика **`new-topic`**.

```
GROUP           TOPIC           PARTITION  CURRENT-OFFSET  LOG-END-OFFSET  LAG             CONSUMER-ID
HOST            CLIENT-ID
new-group       new-topic       0          3               3               0               consumer-new-group-1-a9f82106-f111-
4a54-a828-4a418b17ffa 720 92.10.38      consumer-new-group-1
```

Назначение топикам групп потребителей

Каждая создаваемая клиентом каждый группа потребителей, использующий на топик или партицию, из которых потребляется топик группы потребителя. Точно обрабатывает потребителя, назначенного группе потребителя, и группа потребителей получит сообщение от потребителя в партиции.

Если необходимо передать сообщение нескольким потребителям, следует назначить их разные группы потребителя.

Пример назначения группы потребителю

Необходимо во время подключения к топику использовать опцию **`--group`**

```
$ /usr/lib/kafka/bin/kafka-console-consumer.sh --topic new-topic --from-beginning --group new-group --bootstrap-server
hostname:9092
```

Настройка безопасной среды

Для получения информации о настройке безопасности в среде Kafka можно перейти в раздел **Базовые операции** → **Kafka** → **Управление доступом**.

Аппаратное обеспечение

Общие рекомендации при выборе оборудования

- При возможности использовать для брокеров Kafka отдельные виртуальные или стойки.
- Для применения оптимального конфигурации диски в системе не менее 3 брокеров Kafka.
- Обеспечить для high-availability в соответствии, описанными в статье, аппаратные требования к серверам согласно планируемой нагрузке.

Рекомендации при выборе памяти

Kafka при работе может нуждаться в большом объеме файловой системы для хранения сообщений и может использует текстопроцессоры.

Kafka при работе может нуждаться в большом объеме файловой системы для хранения сообщений и может использует текстопроцессоры.

Рекомендации при выборе процессоров

- Включение защиты канала по протоколу SSL может увеличить требования к процессору (точные данные зависят от типа ЦП и реализации JVM).
- При выборе процессоров между более быстрым процессором с меньшим количеством ядер лучше выбрать большее количество ядер. Дополнительные параллельные, которые предлагает несколько ядер, предпочтительнее более высокой скорости.

Хранение данных

- Не рекомендуется использовать massive data pools. JBOD, так как для корректного функционирования требуется одна точка монтирования.
- Рекомендуется использовать несколько дисков, поскольку максимальную пропускную способность.
- Не рекомендуется использовать один и то же диски, используемые для данных Kafka, совместно с журналами приложений или другой активностью файловой системы или ОС.
- Если вы планируете использовать несколько каталогов данных, брокер может перенос партиции после с наименьшим количеством сохраненных партиций. Каждая партиция будет размещена в другом каталоге, а не в том, который имеет наибольшее количество свободного места. Этот алгоритм не учитывает и распределяет нагрузки между каталогами данных.
- Массив данных RAID 10 рекомендуется как лучший вариант для большинства сегментов использования. Он обеспечивает улучшенную производительность чтения и записи, а также обеспечивает защиту данных, если диск в массиве вышел из строя.
- Основная недостаток RAID заключается в том, что он уменьшает доступное дисковое пространство. Для RAID 10 количество включенных дисков уменьшается вдвое при использовании массива при сбое диска. Стоимость восстановления отключенного RAID-массива с учетом нагрузки может быть значительной.
- Не рекомендуется использовать виртуальные хранилища Network Attached Storage (NAS). NAS часто медленнее, подвержен большим задержкам с более широким отклонением средней задержки и является одной точкой отказа.

Сеть

Требования к сетевым подключениям приведены в статье, аппаратные требования сети. Современные сети сегодня могут легко обеспечивать данные, а высокая пропускная способность помогает передавать и распределять данные между брокерами. Современные сети сегодня обработки данных (1 GbE, 10 GbE) достаточны для подключения большинства систем.

Программное обеспечение

Общие рекомендации

- Требования к программному обеспечению описаны в статье, Программные требования.
- Для запуска Kafka рекомендуется использовать среду выполнения Java версии 8 или выше.
- Не рекомендуется использовать версии Java, чтобы убедиться, что они получили безопасности устранены.
- Рекомендованная настройка GC (приведенная на больших развертываниях с JDK 1.8.0) выглядит следующим образом:

```
-Xmx6g -Xms6g -XX:MetaspaceSize=96m -XX:+UseG1GC -XX:MaxGCPauseMillis=20
       -XX:InitiatingHeapOccupancyPercent=35 -XX:G1HeapRegionSize=16M
       -XX:MinMetaspaceFreeRatio=50 -XX:MaxMetaspaceFreeRatio=80
```

- Рекомендуем также рекомендации могут быть улучшения совместимости между Kafka и новой операционной системой. Для получения применения рекомендаций необходимо обратиться к документации вашей операционной системы.

Файловые дескрипторы

Kafka в любой момент потребуется может требовать относительно большого количества дескрипторов файловых дескрипторов.

Количество дескрипторов файловых дескрипторов, которое может потребоваться ОС для работы брокера, увеличивается для каждого процесса. Это недостаточно для Kafka.

Рекомендуется увеличить ограничение файловых дескрипторов для значения по умолчанию, в зависимости, и нижнего больше.

Для увеличения в Kafka, как и для любого процесса, ограничение файловых дескрипторов — `ulimit -n` файл, устанавливающий имеет доступа для операционной системы через PAM (Pluggable Access Management Files). Для этого требуется для всех пользователей увеличить значение параметра **`nofile`** максимальное число открытых дескрипторов. Для этого **`hard`** и **`soft`**, добавив в эти настройки limits.conf строки:

```
* hard nofile 100000
* soft nofile 100000
```

Виртуальная память

Может потребоваться изменить чрезмерный размер виртуальной памяти **`vm.max_map_count`**, позволяющий обрабатывать необходимое количество файлов mmap.

Kafka использует файлы индексов mmap, необходимо узнать значение этого параметра для Kafka и которого должно Kafka. Файлы Jidex предотвращает особой большого файлов с отображением памяти.

Подсчитать файлы Jidex можно при помощи этой команды:

```
$ find . -name '*index' | wc -l
```

Установить **`vm.max_map_count`** в файле /etc/sysctl.conf для этого. Это установит текущие количество отображаемых в память файлов. Максимальное значение памяти **`65530`**. **`vm.max_map_count`** — это количество открытых файлов **`65530`**.

Чтобы узнать, сколько дескрипторов файлов Jidex, чтобы узнать для сегмента формула Jidex, чтобы узнать для сегмента формула:

```
$ sysctl -w vm.max_map_count=262144
$ echo "vm.max_map_count=262144" >> /etc/sysctl.conf
$ sysctl -p
```

Рекомендации по настройке NiFi

Ольга Семме

Содержание

Назад к содержанию

Увеличение количества файловых дескрипторов

Увеличение количества разветвленных процессов

Увеличение количества доступных портов сокетов TCP

Установка длительности состояния TIMED_WAIT для сокетов

Отключение параметра подкачки

Увеличение ресурсов аппаратного обеспечения

Типовые настройки для Linux могут не подходить для приложений с интенсивным вводом-выводом, таких как NiFi. Для используемых областей операционной системы требования дистрибутива могут различаться.

Приведенные ниже рекомендации могут быть полезны для улучшения совместимости между NiFi и вашей операционной системой.

Для лучшего применения рекомендаций необходимо обратиться к документации по конкретному дистрибутиву.

Увеличение количества файловых дескрипторов

NiFi в любой момент потенциально может иметь очень большое количество открытых файловых дескрипторов.

Дескрипторы файлов — это целочисленные идентификаторы, определяющие структуры данных. Ядро Linux обращается к этим структурам как к файловым структурам, поскольку они описывают открытые файлы. Индекс файловой структуры является дескриптором файла.

Для изменения лимита открытых файлов необходимо отредактировать */etc/security/limits.conf* — файл, устанавливающий лимиты ресурсов для пользователей, вошедших в систему через PAMPrivileged Access Management (PAM). Для этого требуется для всех групп пользователей увеличить значение параметра `nofile` — максимальное число открытых файлов для типов `hard` и `soft`, добавив в */etc/security/limits.conf* строки:

```
*  hard  nofile  50000
*  soft  nofile  50000
```

Необходимо при увеличении лимита учитывать размер памяти ядра операционной системы.

Увеличение количества разветвленных процессов

NiFi может быть настроен на создание значительного количества потоков. Чтобы увеличить допустимое число, также необходимо отредактировать файл */etc/security/limits.conf*. Для этого требуется для всех групп пользователей увеличить значение параметра `nproc` — максимальное число процессов для типов `hard` и `soft`, добавив в */etc/security/limits.conf* строки:

```
*  hard  nproc  10000
*  soft  nproc  10000
```

И также при наличии в вашей операционной системе необходимо отредактировать файл */etc/security/limits.d/90-nproc.conf* или */etc/security/limits.d/20-nproc.conf*, переопределяющий значение `nproc`, заданное в файле */etc/security/limits.conf*. В этом файле необходимо для всех групп пользователей увеличить значение параметра `nproc` — максимальное число процессов для типа `soft`, добавив строку:

```
*  soft  nproc  10000
```

Увеличение количества доступных портов сокетов TCP

Увеличение количества доступных портов сокетов TCP особенно важно, если ваш поток будет устанавливать и отключать большое количество сокетов за небольшой период времени.

Для установки максимального количества сокетов TCP необходимо выполнить команду:

```
$ sudo sysctl -w net.ipv4.ip_local_port_range="10000 65000"
```

Установка длительности состояния TIMED_WAIT для сокетов

TIME_WAIT — это состояние серверного сокета, в котором он выдерживает тайм-аут, чтобы собирать случайно задержавшиеся в сети пакеты. Длительность тайм-аута можно изменять для получения возможности быстро настраивать и отключать новые сокеты.

Для изменения длительности тайм-аута используются следующие команды:

- для дистрибутивов с ядром 2.6:

```
$ sudo sysctl -w net.ipv4.netfilter.ip_conntrack_tcp_timeout_time_wait="1"
```

- для дистрибутивов с ядром 3.0:

```
$ sudo sysctl -w net.netfilter.nf_conntrack_tcp_timeout_time_wait="1"
```

Отключение параметра подкачки

Параметр подкачки (swap) влияет на предпочтение ядра перемещать страницы памяти из приложений на страницу подкачки, а не высвобождать память из кеша.

Для улучшения эффективности NiFi рекомендуется установить параметр `swappiness` равным нулю. Это гарантирует, что если память ограничена, кеш страниц будет уменьшен в попытке восстановить память до того, как страницы приложения будут перемещены в пространство подкачки.

Чтобы отключить параметр подкачки, необходимо отредактировать */etc/sysctl.conf* — файл, предназначенный для управления параметрами ядра.

Необходимо добавить следующую строку:

```
vm.swappiness = 0
```

Для разделов, обрабатывающих различные репозитории NiFi, отключите параметр `atime` — поле для записи метки времени. Это может вызвать неожиданный скачок пропускной способности. Для этого в файле */etc/fstab* (который используется для настройки параметров монтирования различных блочных устройств, разделов на диске и удаленных файловых систем) необходимо добавить для интересующих разделов параметр `noatime`, отключающий полностью запись времени доступа к файлу.

Увеличение ресурсов аппаратного обеспечения

Для эффективной работы NiFi рекомендуется:

- При возможности использовать для экземпляров NiFi отдельные сервера или стойки.
- Обеспечить HA (High Availability) в соответствии с требованиями, описанными в статье Аппаратные требования к серверам.

Сценарии использования

Ольга Семке

Содержание

Назад к содержанию

Обмен сообщениями

Отслеживание активности на сайте

Метрики

Агрегация журналов

Потоковая обработка

Event sourcing

Commit log

Статья описывает сценарии использования ADS.

Обмен сообщениями

Для оптимальной организации обмена **сообщениями** (**messaging**) используются **брокеры сообщений** — приложения, которые выступают посредниками между приложениями источника и приемника, использующих разные протоколы. Kafka обладает несколькими преимуществами по сравнению с большинством брокеров сообщений:

- Высокая **пропускная способность** позволяет обработать большой объем данных за незначительный период времени.
- Организованное встроенное **партиционирование** (**partitioning**) позволяет контролировать время хранения и объем данных за счет того, что сообщения разделены на сегменты по партициям и не хранятся в одном большом файле.
- **Репликация** позволяет автоматически переключаться на реплики записей при отказе сервера в кластере, поэтому сообщения остаются доступными даже при сбоях.
- **Отказоустойчивость** гарантирует, что система обмена сообщениями будет сохранять сообщения при отсутствии активных подписчиков до тех пор, пока новый потребитель не подпишется на очередь сообщений.

Сервисы, дополняющие и оптимизирующие работу Kafka

Сервис	Сценарий взаимодействия с Kafka	Преимущества использования
NiFi	NiFi как **производитель** (**producer**) — принимает данные из источников непосредственно в центральный экземпляр NiFi, который доставляет данные в соответствующий топик Kafka	▪ Передача данных в Kafka без специального ПО. ▪ Возможность визуально отслеживать и контролировать конвейер данных.
	NiFi как **потребитель** (**consumer**) — принимает данные из Kafka и передает их в другую систему	▪ Передача данных из Kafka в некоторые файловые и иные внешние системы (HFDS, HBase) без специального ПО. ▪ Возможность визуально отслеживать и контролировать конвейер данных.
	Динамический самонастраивающийся поток данных — объединение возможностей NiFi, Kafka и аналитических платформ: 1. NiFi передает данные в Kafka. 2. Данные становятся доступными для аналитических платформ. 3. Результаты записываются обратно в другой топик Kafka. 4. Данные поступают в NiFi.	▪ Возможность дальнейшего использования данных, полученных NiFi, без специального ПО. ▪ Возможность визуально отслеживать и контролировать конвейер данных.
ZooKeeper	Сообщает брокерам Kafka о текущем состоянии кластера	Автоматическое обновление метаданных у клиента Kafka при подключении к любому брокеру
Confluent Schema Registry]	Реализует механизм работы с разными схемами данных Kafka	▪ Определение полей, необходимых для описания события. ▪ Документирование событий и значений каждого его поля в доступном виде. ▪ Предотвращение искажения данных. ▪ Работа с различными форматами данных от нескольких источников информации.

Отслеживание активности на сайте

Kafka — отличный инструмент для отслеживания **активности на веб-сайте** (**website activity tracking**). Когда новый пользователь регистрируется на веб-сайте, его активность может отслеживаться следующим образом:

1. Пользователь нажимает кнопку (элемент UI) в интерфейсе веб-страницы.
2. Веб-приложение создает сообщение с метаданными данного элемента UI.
3. Сообщения с метаданными собираются в пакете данных и отправляются в Kafka, создавая при этом **логи фиксации** (**commit log**).
4. При следующем действии пользователя с этим элементом UI сообщение о действии добавляется в логи фиксации, а смещение сообщения в очереди увеличивается.

Далее из Kafka эти данные могут быть собраны для аналитики. Эти данные показывают использование веб-сайта в режиме реального времени.

Метрики

Kafka может использоваться для оперативного сбора **метрик** (**metrics**) различных приложений и операционных данных:

- технологические процессы;
- статистика аудита и сбора данных;
- активность системы;
- агрегирование статистики работающих приложений и инфраструктуры;
- трекеры потребления потоков данных пользователями в режиме реального времени.

Kafka обладает несколькими преимуществами при использовании в мониторинге:

- Возможность подключать новых производителей для отправки метрик, а затем использовать данные мониторинга для нескольких различных систем.
- Возможность выполнять анализ в реальном времени по большому набору данных, одновременно со сбором метрик.
- Возможность использовать небольшой объем кода для обработки данных в приложении потребителя.
- Возможность выделить в разные модули данные в соответствии с их назначением.

Агрегация журналов

Агрегация журналов (**log aggregation**) — один из возможных сценариев использования ADS. Для распределенных сред с разными архитектурами Kafka является наиболее оптимальным средством для агрегации журналов. Kafka собирает файлы журналов с серверов и помещает их в центральное хранилище (возможно, на файловый сервер или HDFS) для обработки. По сравнению с другими системами Kafka обладает рядом преимуществ:

- Kafka абстрагируется от деталей файлов и представляет абстракцию для данных журнала или событий в виде потока сообщений. Это обеспечивает обработку с меньшей задержкой и упрощает поддержку нескольких источников данных и распределенного потребления данных.
- Kafka предлагает столь же высокую производительность, как и другие системы, но более высокие гарантии отказоустойчивости и гораздо меньшую задержку в обработке данных.

Потоковая обработка

Для улучшения производительности обмена сообщениями используется **потоковая обработка данных** (**stream processing**). Преимущества потоковой передачи данных с использованием Kafka перед пакетными конвейерами:

- Потоки сообщений обрабатываются в реальном времени.
- К сообщениям применяются операции преобразования, фильтрации, агрегирования или соединения, чтобы опубликовать обработанные сообщения в другом потоке.
- Потоковые конвейеры снижают нагрузку на источник данных, т.к. не нужно выполнять полные запросы, а можно извлекать данные из лог-файлов СУБД или других систем.
- Сообщения могут храниться столько, сколько необходимо.

Инструменты и сервисы для потоковой передачи данных

Инструмент или сервис	Роль в сценарии потоковой обработки	Преимущества использования
Kafka Streams	Клиентская библиотека для разработки потоковых приложений Big Data, которые работают с данными, хранящимися в топиках Kafka	Мощный и гибкий API-интерфейс со всеми преимуществами Kafka (масштабируемость, надежность, минимальная задержка, механизмы аналитических запросов) позволяет разработчику писать код в локальном режиме (вне кластера)
NiFi	Использует концепцию потока как последовательность операций: передача, преобразование и обогащение данных над последовательностью отдельных **событий** (**events**)	Поток НЕ рассматривается как большая **пакетная** (**batch**) операция, требующая выполнения первоначальной загрузки всех данных перед началом обработки. Например, база данных SQL с миллионами строк рассматривается как миллионы отдельных строк, требующих своей обработки
ksqlDB	Платформа для потоковой обработки больших данных в Kafka с помощью структурированных SQL-запросов	Структуры данных в ksqlDB — это программные единицы, которые способны хранить и обрабатывать множество данных, связанных логикой ksqlDB

Event sourcing

Kafka является оптимальным вариантом для поддержки приложений, созданных по шаблону **event sourcing** для обработки событий. Преимущества использования Kafka в event sourcing:

- Поддержка хранения данных журналов значительного объема.
- Возможность работы с отложенными событиями благодаря механизмам очереди и смещений в Kafka.

Commit log

Kafka может служить внешним commit log для распределенной системы. Преимущества использования Kafka:

- Возможность репликации данных между узлами.
- Возможность восстановления данных (механизм повторной синхронизации для отказавших узлов).
- Функция сжатия журналов в Kafka.

Архитектура ADS

Ольга Семме

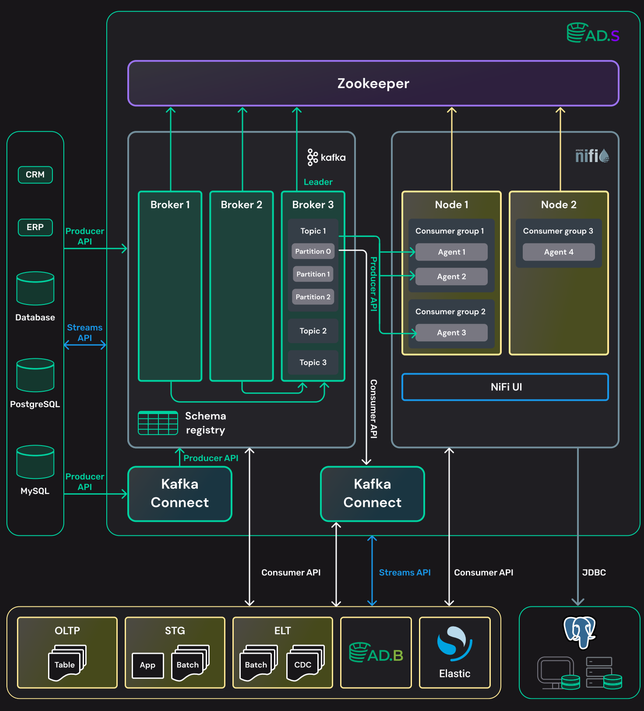
Содержание

Назад к содержанию

- Обзор
  - Kafka
  - ZooKeeper
  - NiFi
  - API

Обзор

Для каждого случая использования архитектура ADS определяется пользователем в зависимости от поставленных задач. Верхний уровень одного из вариантов архитектуры ADS показан на рисунке.



Архитектура ADS

Kafka

- Каждый из именованных **топиков** (**topic**) в Kafka состоит из одной и более **партиций** (**partition**), распределенных между **брокерами** (**broker**) внутри одного кластера ADS.
- Когда новое сообщение добавляется в топик, оно записывается в одну из партиций этого топика. Сообщения с одинаковыми ключами всегда записываются в одну и ту же партицию, тем самым гарантируя очередность или порядок записи и чтения.
- Для гарантии сохранности данных каждая партиция в Kafka может быть реплицирована несколько раз. Таким образом гарантируется наличие нескольких копий сообщения, хранящихся на разных брокерах.
- У каждой партиции есть **лидер** (**leader**) — брокер, который работает с клиентами. Именно лидер работает с **производителями** (**producer**) и в общем случае отдает сообщения **потребителям** (**consumer**). Сообщения всегда отправляются лидеру и, в общем случае, читаются с лидера.
- Для масштабируемой и надежной потоковой передачи данных между Kafka и другими системами данных используется Kafka Connect. Kafka Connect может принимать целые базы данных или собирать метрики с серверов приложений в топики Kafka, а также может доставлять данные из разделов Kafka, например, в системы для анализа и поиска данных (ADLS) или в системы управления базами данных (ADB).
- При помощи Confluent Schema Registry обеспечивается совместимость схем данных между производителем и потребителем Kafka.

ZooKeeper

- Решение о том, какой из брокеров является контроллером (то есть брокером, отвечающим за выбор лидеров партиций) принимает ZooKeeper. Также ZooKeeper определяет, в каком состоянии находятся лидеры партиций и их реплики.
- Дополнительно ZooKeeper выполняет роль консистентного хранилища метаданных и распределенного сервиса логов. В случае сбоя брокера именно в ZooKeeper контроллером будет записана информация о новых лидерах партиций.

NiFi

- Для работы с потоковыми файлами NiFi использует множество обработчиков (процессоров) – отдельных фрагментов кода для выполнения конкретной операции с потоковыми файлами.
- Каждый потребитель Kafka является частью **Consumer Group**, которая настроена при помощи процессора ConsumeKafka в NiFi на конкретный брокер и имя группы Kafka.
- Каждая Consumer Group имеет уникальное название и регистрируется брокерами в кластере Kafka. Данные из одного и того же топика могут считываться множеством Consumer Group одновременно.
- Когда несколько потребителей читают данные из Kafka и являются членами одной и той же группы, то каждый из них получает сообщения из разных партиций топика, таким образом распределяя нагрузку. NiFi в данном случае выступает в роли потребителя Kafka и реализует всю логику обработки данных.
- Также существует возможность при помощи NiFi передать данные из Kafka в другую систему.
- Веб-интерфейс пользователя NiFi позволяет осуществлять разработку, управление и мониторинг в едином интерфейсе.

API

- **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 **потокового процессора** (**stream processor**). 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.

Концепции хранения в Kafka

Ольга Сенина

Содержание

Назад к содержанию

Основные компоненты Kafka
Основные параметры при работе с Kafka
Последовательность записи и чтения сообщений
Репликация и распределение партиций в брокере
- Репликация
- Синхронизация реплик
- Брокеры

Группы потребителей
- Точка-точка
- Публикация-подписка

Основные компоненты Kafka

Концепция хранения в Kafka основана на следующих компонентах:

- **Сообщение (message)** — запись произвольного события, в том числе состояние объекта, значения физической величины или любого другого параметра, который требует отслеживания, хранения или передачи в другую систему.
- **Производитель (producer)** — клиентское приложение, публикующее (записывающее) сообщения в Kafka.
- **Потребитель (consumer)** — клиентское приложение, которое подписывается (читает и обрабатывает) сообщения.
- **Топик (topic)** — категория (тема), в которой сообщения публикуются в Kafka. В топике сообщения пишутся в журнал коммитов.
- **Журнал коммитов (commit log)** — упорядоченная структура сообщений, доступная только для записи. В журнале коммитов нельзя ни изменить, ни удалить данные. Большие журналы разделяют на партиции. Если журнал состоит из одной партиции, то он и является партицией.
- **Партиция (partition)** — упорядоченная неизменяемая последовательность записей, содержимое журнала коммитов.
- **Брокер (broker)** — сервер, на котором хранятся партиции.
- **Сегмент (segment)** — часть партиции, хранящаяся в отдельном файле на диске.
- **Группа потребителей (consumer group)** — несколько потребителей, объединенных в группу для получения сообщений из одного или нескольких топиков. Партиции одного топика автоматически равномерно распределяются между потребителями в одной группе. Каждая партиция доступна только одному потребителю из группы.

Основные параметры при работе с Kafka

Производитель при записи сообщений указывает:

- топик, в которой будут записаны данные;
- партицию, в которую будут записаны сообщения, имеющие одинаковый ключ (если это необходимо для структурирования данных в топике);
- срок хранения сообщений в Kafka;
- необходимое сжатие;
- необходимый уровень подтверждения о записи сообщения в партицию и ее реплик;
- коэффициент репликации партиций.

Потребитель для чтения сообщений указывает:

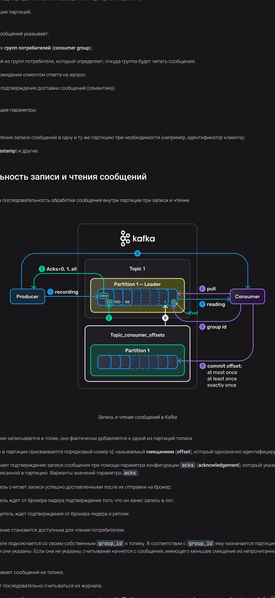
- количество уникальных **групп потребителей (consumer group)**;
- `group_id` для каждой из групп потребителя, который определяет, откуда группа будет читать сообщения;
- максимальное время ожидания клиентом ответа на запрос;
- необходимый уровень подтверждения доставки сообщений (семантика).

Сообщение имеет следующие параметры:

- **данные (data)**;
- **ключ (key)**, для определения записи сообщений в одну и ту же партицию при необходимости (например, идентификатор клиента);
- **отметка времени (timestamp)** и другие.

Последовательность записи и чтения сообщений

На рисунке ниже показана последовательность обработки сообщения внутри партиции при записи и чтении.



Запись и чтение сообщений в Kafka

1. Когда новое сообщение записывается в топик, оно фактически добавляется к одной из партиций топика.
2. Каждому сообщению в партиции присваивается порядковый номер id, называемый **смещением (offset)**, который однозначно идентифицирует каждую запись.
3. Производитель получает подтверждение записи сообщения при помощи параметра конфигурации `acks` (**acknowledgement**), который указывает при каком условии запись считается записанной в партицию. Варианты значений параметра `acks`:
   - `0` — производитель считает запись успешно доставленным после их отправки на брокер;
   - `1` — производитель ждет от брокера лидера подтверждение того, что он записс записал в лог;
   - `all` — производитель ждет подтверждения от брокера лидера и реплик.
4. Далее данное сообщение становится доступны для чтения потребителем.
5. Экземпляр потребителя подключается со своим собственным `group_id` к топику. В соответствии с `group_id` ему назначается партиция для чтения и смещение в этой партиции, если они указаны. Если они не указаны, считывание начинается с сообщения, имеющего меньшее смещение из непрочитанных (самое ранее из записанных).
6. Потребитель запрашивает сообщения из топика.
7. Сообщения начинают последовательно считываться из журнала.
8. При чтении из партиции потребитель делает **коммит смещения (commit offset)** - запись о считанном сообщении в служебный топик Kafka `__consumer_offsets`. Коммит смещения выполняется в соответствии с семантикой доставки сообщения, настроенной в приложении потребителя. Могут быть обеспечены следующие семантики доставки:
   - at most once — смещение применяется сразу после получения сообщения (если обработка не происходит, то сообщение теряется);
   - at least once — смещение применяется только после обработки сообщения (если обработка не происходит, то сообщение считывается повторно);
   - exactly once — все сообщения поставляются один раз.
9. Сообщение со следующим по счету смещением становится доступно для чтения.

Метаданные каждого сообщения являются его смещением или положением в журнале. Смещение для начала чтения сообщения может быть установлено самим потребителем:

- линейное продвижение смещения при считывании записи (по умолчанию);
- считывание записи в любом порядке при необходимости (возвращение к более старому смещению или переход к самой последней актуальной записи).

Репликация и распределение партиций в брокере

Репликация

Чтобы сделать сообщения отказоустойчивыми и высокодоступными, каждый топик с входящими в него партициями можно **реплицировать**. Это значит создать копию и поместить ее на сервер (записать на диск).

Коэффициент репликации (replication factor) — параметр, определяющий, сколько копий каждой партиции будет распределено на брокерах. Рекомендуемым значением коэффициента репликации (значение по умолчанию) является `3`.

Каждая реплика партиции записана на отдельном брокере. Это значит, что при сбое или отказе брокера, на котором находится одна из реплик, останется еще две копии данной партиции. Это обеспечивает постоянную доступность данных даже при технических обслуживании брокеров.

Брокеры разбивают каждую партицию на **сегменты (segment)**, каждый из которых хранится в отдельном файле данных на диске, подключенном к брокеру. По умолчанию каждый сегмент содержит либо 1 ГБ данных, либо данные за неделю, в зависимости от того, какой предел будет достигнут первым. При достижении лимита сегмента брокер закрывает файл и создает новый файл для записи новых сообщений. **Активный сегмент** — файл, в который в данный момент времени производится запись. **Закрытый сегмент** — файл, в котором хранятся сообщения и запись новых сообщений не производится.

Синхронизация реплик

Формат репликации называется **In/Sync**. Это значит, что из всех реплик одна является **лидером (leader)**, а другие **последователями (followers)**. Производитель может управлять синхронностью/асинхронностью репликации при помощи параметра `acks` (**acknowledgement**).

Запись и чтение сообщений осуществляется через ведущую-репликну партицию. Последователь, содержащий все сообщения, хранящиеся у лидера, называется синхронизированной репликой (**in-sync replica**, **ISR**).

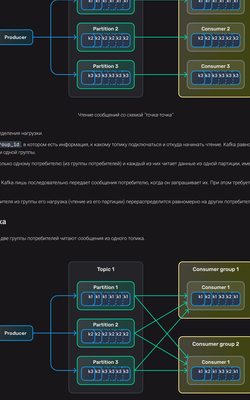
Брокеры

Для обеспечения отказоустойчивости в кластере Kafka лидеры партиций автоматически распределяются по разным брокерам. Каждый брокер может быть лидером для одних партиций и последователем для других.

Если брокер, выступающий в качестве лидера для партиции, отключается — любой брокер, который находится в актуализированном или синхронизированном состоянии для этой партиции, может взять на себя роль лидера.

На рисунке ниже показано, как могут быть распределены партиции с заданной конфигурацией кластера:

- количество брокеров — 3;
- количество топиков — 2;
- количество партиций в `Topic 1` — 2;
- количество партиций в `Topic 2` — 1;
- фактор репликации — 3.



Распределение партиций между брокерами Kafka

Группы потребителей

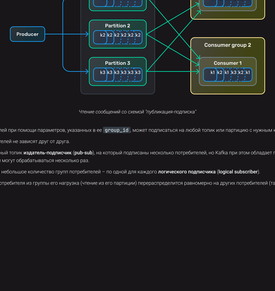
Возможно создать большое количество конфигураций групп потребителей для чтения из топиков Kafka, но все они основаны на двух основных схемах обмена сообщениями — **точка-точка (point-to-point)** и **публикация-подписка (publish-subscribe)**.

Группы потребителей управляются **координатором группы потребителей**.

Координатор группы потребителей — один из брокеров, который делает опрос всех потребителей группы потребителей. Если потребитель перестает отправлять heartbeats, координатор инициирует ребалансировку потребителей.

Точка-точка

На рисунке ниже показано, как одна группа потребителей читает сообщения из одного топика.



Чтение сообщений со схемой "точка-точка"

Это вариант идеального распределения нагрузки.

У группы потребителей есть `group_id`, в котором есть информация, к какому топику подключаться и откуда начинать чтение. Kafka равномерно распределяет все партиции между потребителями одной группы.

Kafka дает доступ к партиции только одному потребителю (из группы потребителей) и каждый из них читает данные из одной партиции, имеющей одинаковый ключ (k1, k2, k3).

В данной конфигурации брокер Kafka последовательно передает сообщения потребителю, когда он запрашивает их. При этом требуется минимум дополнительных настроек.

При отключении одного потребителя из группы его нагрузка (чтение из его партиции) перераспределяется равномерно на другие потребителей.

Публикация-подписка

На рисунке ниже показано, как две группы потребителей читают сообщения из одного топика.

Чтение сообщений со схемой "публикация-подписка"

Каждая группа потребителей при помощи параметров, указанных в их `group_id`, может подписаться на любой топик или партицию с нужным ключом (k1, k2, k3).

При этом группы потребителей не зависят друг от друга.

Kafka действует как обычный топик **издатель-подписчик (pub-sub)**, на который подписаны несколько потребителей, но Kafka при этом обладает преимуществом — все сообщения сохраняются и могут обрабатываться несколько раз.

Чаще всего топики имеют небольшое количество групп потребителей — по одной для каждого **логического подписчика (logical subscriber)**.

При отключении одного потребителя из группы его нагрузка (чтение из его партиции) перераспределяется равномерно на другие потребителей (так же как и в схеме «точка-точка»).

Сжатие журналов в Kafka

© Data Egret

Содержание

Назад к содержанию

Политика очищения журналов
Удаление (delete)
Сжатие (compact)
- Обзор политики сжатия
- Запуск сжатия
- Алгоритм сжатия
- Удаление при помощи сжатия
- Гарантии и преимущества сжатия

Удаление и сжатие (delete and compact)
Настройка сжатия
- Настройка кластера
- Настройка топика

В данной статье описаны существующие политики очищения (сжатия) журналов в Kafka, параметры, используемые при настройке политик, а также способы настройки политик.

Политика очищения журналов

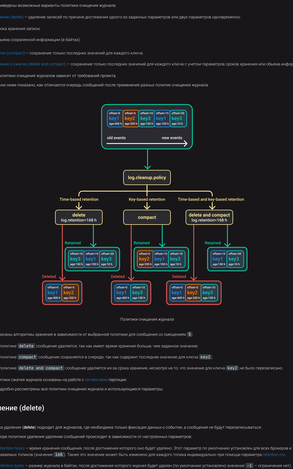
Политика очищения журнала (`log.cleanup.policy`) – основной параметр, определяющий стратегию удаления старых записей.

Ниже приведены возможные варианты политики очищения журнала:

- Удаление (delete) – удаление записей по прочиле достижения одного из заданных параметров или двух параметров одновременно:
  - срока хранения записи;
  - объема сохраняемой информации (в байтах).
- Сжатие (compact) – сохранение только последних значений для каждого ключа.
- Удаление и сжатие (delete and compact) – сохранение только последних значений для каждого ключа с учетом параметров срока хранения или объема информации.

Выбор политики очищения журналов зависит от требований проекта.

На рисунке ниже показано, как отличаются между собой доступные после применения разных политик очищения журнала.



Политики очищения журнала

Ниже описаны алгоритмы хранения в зависимости от выбранной политики для сообщения со значением `5`:

- при политике `delete` сообщение удалится, так как имеет верхнее хранения более, чем заданное значение;
- при политике `compact` сообщение сохранится в очереди, так как содержит последнее значение для ключа `key2`;
- при политике `delete and compact` сообщение также сохранится, несмотря на то, что значение для ключа `key2` не было перезаписано.

Все политики имеют нюансы при работе в отличительных партициях.

Ниже подробно рассмотрены все политики очищения журнала и используемые параметры.

Удаление (delete)

Политика удаления **delete**, подходит для журналов, где необходимы только фиксация данных о событии, а сообщения не будут перезагружаться.

При выборе политики удаления сообщений происходит в зависимости от настроенных параметров:

- `log.retention.hours` – время хранения сообщения, после достижения которого оно будет удалено. Этот параметр по умолчанию установлен для всех брокеров и создаваемых топиков значением `168`. Также эта значение может быть изменено для каждого топика индивидуально при помощи параметра `retention.ms`.
- `log.retention.bytes` – размер журнала в байтах, после достижения которого журнал будет удален (по умолчанию установлено значение `-1` – ограничения нет). Также это значение может быть установлено для топика индивидуально при помощи параметра `retention.bytes`.

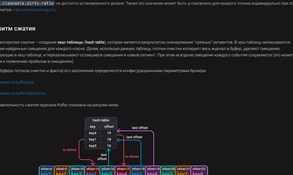
Также допускается устанавливать сразу оба параметра для одновременного управления временем хранения сообщений и объемом журнала.

Для указания частоты удаления сообщений используется параметр `log.cleanup.interval.mins`, определяющий, как часто средство очистки журналов проверяет журналы, подлежащие удалению.

Для контроля удаления сообщений рассматриваются следующие виды сегментов партиции журнала:

- Закрытые сегменты журнала **log segment** – сегменты, в которых уже записываются данные;
- Активный сегмент журнала **active log segment** – сегмент, в который в данный момент времени производится запись. Активный сегмент не участвует в алгоритме удаления сообщений.

Схема удаления сообщений журнала Kafka при ограничении по времени хранения показана на рисунке ниже.



Удаление сообщений

Сообщения удаляются в составе сегментов в следующей последовательности:

1. Определяется сообщение с максимальной временной меткой (самое новое из записанных) внутри одного сегмента.
2. Определяется значение времени записи сообщения с временной меткой (`Time`) и текущим временем (`Tcurrent`), сравнение полученного значения со значением, хранимым в `log.retention.hours` (`retention.ms` для топиков).
3. Если для сегмента выполняется условие `Tcurrent - Time > log.retention`, то сегмент удаляется целиком.

Таким образом, общее время жизни сообщений может быть больше указанного значения в `log.retention.hours` и зависит от установленного размера журнала сегмента `segment.bytes`.

Сжатие (compact)

Обзор политики сжатия

Политика **сжатия (compact)** обычно применяется для журналов, содержание в которых имеет формат ключ/значение. Например, базы данных, сохраняющие часто изменяющиеся значения для одних и тех же полей.

Эта политика используется не для хранения всех значений, а только для простого хранения последних значений.

В алгоритме сжатия журналов различаются следующие виды сегментов партиции:

- **Чистый сегмент журнала (clean log segment)** – сегмент, в которых нет повторяющихся ключей, т.е. сегмент содержит только по одному значению для каждого ключа – последнему на момент создания сегмента. В "чистом" сегменте могут быть пробелы в нумерации смещений из-за предыдущего сжатия.
- **Грязный сегмент журнала (dirty log segment)** – каждый сегмент, который был записан после последнего "чистого" сегмента. В нем могут быть повторяющиеся ключи, в таком случае, присутствуют и "чистые" сегменты. "Грязные" сегменты начинаются с точки **cleaner point**.
- **Активный журнал (active log segment)** – сегмент, в который в данный момент времени производится запись. Активный сегмент не участвует в алгоритме сжатия.

Запуск сжатия

Момент запуска сжатия регулируется параметрами брокера:

- `log.cleaner.min.cleanable.ratio` – соотношение объема "грязных" сегментов к общему объему журнала, при котором начинается запуск сжатия (по умолчанию установлено значение `0.5`). Также это значение может быть задано для топика индивидуально при помощи параметра `min.cleanable.dirty.ratio`.
- `log.cleaner.min.compaction.lag.ms` – минимальное время, в течение которого сообщение будет оставаться в журнале несжатым (по умолчанию задано значение `0` – ограничения нет). Сжатие не начинается, если все сообщения в сегменте моложе заданного значения `min.compaction.lag.ms`, доступно установленное время. Также это значение может быть установлено для каждого топика индивидуально при помощи параметра `min.compaction.lag.ms`.
- `log.cleaner.max.compaction.lag.ms` – максимальное время, в течение которого сообщение не подлежит сжатию в журнале (по умолчанию для всех создаваемых топиков задано значение `9223372036854775807`). Также это значение может быть установлено для каждого топика индивидуально при помощи параметра `max.compaction.lag.ms`.

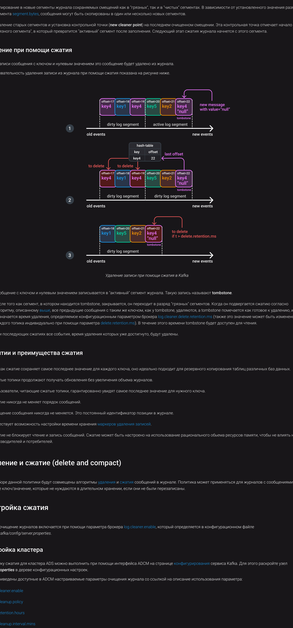
Алгоритм сжатия

Основа алгоритма сжатия – создание **key-таблицы (hash table)**, которая является результатом сканирования "грязных" сегментов. В key-таблицу записываются последние найденные смещения для каждого ключа. Далее, используя данную таблицу, "чистый" сегмент весь журнал в буфер, удаляет сообщения, устаревшие в key-таблице, и перезаписывает "чистый" сегмент. Таким образом, "грязный" сегмент превращается в "чистый" (при этом занятое пространство может уменьшиться). Затем процесс повторяется с следующим "грязным" сегментом.

Размер буфера потока очистки и фактор его заполнения определяются конфигурационными параметрами брокера:

- `log.cleaner.io.buffer.size`
- `log.cleaner.io.buffer.load.factor`

Последовательность сжатия журнала Kafka показана на рисунке ниже.



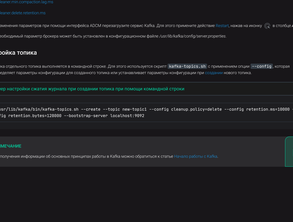
Сжатие журнала

1. Сканирование "грязных сегментов", начиная с точки **cleaner point** и заполнение key-таблицы "активного" сегмента, создание key-таблицы с последними смещениями и определение освобождаемых записей буфер данных. Точка начала сканирования, на шаге интервала, зависит от объема памяти буфера потока очистки.
2. Сканирование "чистых" сегментов и "грязных" сегментов, начиная с точки **cleaner point**, сравнение смещений с key-таблицей, и определение устаревших, которые будут удалены.
3. Копирование в новые сегменты журнала сохраняемых смещений как в "грязных", так и в "чистых" сегментах. В зависимости от установленного значения размера сегмента `segment.bytes`, сообщения могут быть скопированы в один или несколько новых сегментов.
4. Удаление старых сегментов и установка контрольной точки **cleaner point** на последнюю "грязную" сегмента точка (значение начало первого "грязного сегмента", в который предыдущее "активное" сегмента после сканирования. Следующий этап сжатия журнала начнется с этого сегмента.

Удаление при помощи сжатия

После записи сообщения с ключом и нулевым значением это сообщение будет удалено из журнала.

Последовательность удаления записи из журнала при помощи сжатия показана на рисунке ниже.



Удаление записи из журнала при помощи сжатия в Kafka

1. Сообщение с ключом и нулевым значением записывается в "активный" сегмент журнала. Такую запись называют **tombstone**.
2. После того как сегмент, в котором находится tombstone, закрывается, он переходит в разряд "грязных" сегментов. Когда он подвергается сжатию согласно алгоритма, описанному выше, все предыдущие сообщения с таким же ключом, как у tombstone, удаляются, а tombstone помечается как готовое к удалению, а key остается в key-таблице. Определяется конфигурационным параметром брокера `log.cleaner.delete.retention.ms` (также это значение может быть изменено для каждого топика индивидуально при помощи параметра `delete.retention.ms`). В течение этого времени tombstone будет доступен для клиента.
3. При последующем сжатии все сегменты, кроме удаленных которых уже достигнут, будут удалены.

Гарантии и преимущества сжатия

- Так как сжатие сохраняет самое последнее значение для каждого ключа, оно идеально подходит для резервного копирования таблиц различных баз данных.
- Сжатие точно продолжают получать обновления без изменения общих журналов.
- Пользователь читающих сжатые топики, гарантированно увидит самое последнее значение для нужного ключа.
- Сжатие никогда не меняет порядок сообщений.
- Смещение сообщения никогда не меняется. Это постоянный идентификатор позиции в журнале.
- Существует возможность настройки времени удаления маркеров удаления (записей).
- Сжатие не блокирует чтение и запись клиентов. Сжатие может быть настроено на использование рационального объема ресурсов памяти, чтобы не влиять на производительность в потребителях.

Удаление и сжатие (delete and compact)

При выборе данной политики будут совмещены алгоритмы удаления и сжатия сообщений в журнале. Политика может применяться для журналов с сообщениями в формате ключ/значение, которые не нуждаются в длительном хранении, если они не были перезаписаны.

Настройка сжатия

Настройка кластера

В Kafka описаны журналов включается при помощи параметра брокера `log.cleaner.enable`, который определяется в конфигурационном файле /usr/lib/kafka/config/server.properties.

Настройку сжатия для кластера ADS можно выполнить при помощи интерфейса ADCM на странице конфигурирования сервиса Kafka. Для этого разверните узел **server.properties** в дереве конфигурационных параметров.

Ниже приведены доступные в ADCM настраиваемые параметры очищения журналов со ссылкой на описание использования параметра:

- `log.cleaner.enable`
- `log.cleanup.policy`
- `log.retention.hours`
- `log.cleanup.interval.mins`
- `log.cleaner.min.compaction.lag.ms`
- `log.cleaner.delete.retention.ms`

После изменения параметров при помощи интерфейса ADCM перезагрузите сервис Kafka. Для этого примените действие Restart, нажав на иконку 🔄 в действии **Actions**.

Также необходимый параметр брокера может быть установлен в конфигурационном файле /usr/lib/kafka/config/server.properties.

Настройка топика

Настроим отдельного топика выполняется в командной строке. Для этого используется скрипт `kafka-topics.sh` с применением опции `--config`, которая перечисляет параметры конфигурации для создаваемого топика или устанавливает параметры конфигурации при изменении нового топика.

Пример настройки сжатия журнала при создании топика при помощи командной строки

```
[ /usr/lib/kafka/bin/kafka-topics.sh --create --topic new-topic1 --config cleanup.policy=delete --config retention.ms=10000 --
config retention.bytes=128000 --bootstrap-server localhost:9092
```

ПРИМЕЧАНИЕ

Для получения информации об основных принципах работы в Kafka можно обратиться к статье Начало работы с Kafka.

Гарантии доставки сообщений в Kafka

Ольга Сенина

Содержание

Назад к содержанию

- At most once
- At least once
- Exactly once
  - Идемпотентный производитель
  - Транзакции
  - Семантика exactly once

В данной статье описаны гарантии доставки сообщений при использовании семантик доставки, существующих в Kafka.

Основные проблемы доставки сообщений, возникающие при работе с Kafka:

- потеря сообщений при различных сбоях (производителя, потребителя, системы обмена сообщениями, сети);
- дублирование сообщений.

Компоненты Kafka, принимающие участие в последовательности записи и чтении сообщений, могут быть настроены так, чтобы обеспечить следующие семантики доставки:

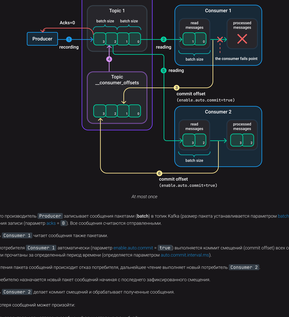
- At most once — сообщения будут обработаны один раз или не будут обработаны вообще (потеряны).
- At least once — сообщения будут обработаны хотя бы один раз.
- Exactly once — каждое сообщение будет обработано один и только один раз.

При использовании разных семантик доставки производительность системы может быть изменена. Более сильная гарантия может снизить производительность, а более слабая гарантия повысит производительность.

At most once

Семантика доставки **at most once** обычно применяется в системах, где предпочтительнее потерять, а не доставить сообщения дважды. Обладает наименьшими "накладными расходами" (дополнительной обработкой сообщений, увеличением скорости обработки, задержки и работы).

Принцип работы семантики доставки at most once показан на рисунке ниже.



At most once

1. Допустим, что производитель `Producer` записывает сообщения пакетами (**batch**) в топик Kafka (размер пакета устанавливается параметром batch.size), не ожидая подтверждения записи (параметр `acks = 0`). Все сообщения считаются отправленными.
2. Потребитель `Consumer 1` читает сообщения также пакетами.
3. Но в случае потребителя `Consumer 1` автоматическая (параметр `enable.auto.commit = true`) выполняется коммит смещений (commit offset) всех сообщений, которые были прочитаны за определенный период времени (определяется параметром auto.commit.interval.ms).
4. Если после чтения пакета сообщений происходит сбой потребителя, дальнейшее чтение выполняет новый потребитель `Consumer 2`.
5. Новому потребителю назначается новый пакет сообщений начиная с последнего зафиксированного смещения.
6. Потребитель `Consumer 2` делает коммит смещений и обрабатывает полученные сообщения.

В данной семье потеря сообщений может произойти:

- на стороне производителя, если отправка сообщений осуществлена с ошибкой;
- на стороне брокера Kafka при сбое брокера
- на стороне потребителя, если коммит смещений (`consumer.commitSync`) произошел не обработанных, задержанных в API класса KafkaConsumer.

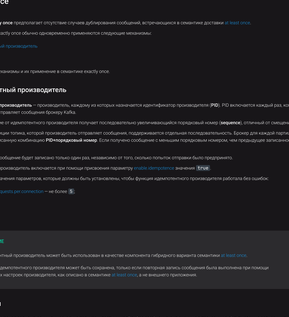
Принцип работы семантики доставки at least once показан на рисунке ниже.

At least once

Семантика доставки **at least once** обычно применяется в системах, где недопустимо терять даже одно сообщение. В случае сбоя некоторые сообщения могут быть обработаны дублированные. Обладает вероятными "накладными расходами" в части работы производителя и брокера (дополнительной обработкой сообщений, увеличение скорости обработки, задержки и работы).

Как правило, по умолчанию реализована семантика автоматический коммит смещений отключает (параметр `enable.auto.commit = false`). При этом используется ручной коммит смещений после обработки сообщений, например, при помощи метода `Consumer.commitSync` (синхронное фиксирование до тех пор, пока брокер не ответит о запуска на фиксацию) или `Consumer.commitAsync` (асинхронное не блокируется, задержка в API класса KafkaConsumer.

Принцип работы семантики доставки at least once показан на рисунке ниже.



At least once

Описание семантики и возможной проблемы на стороне производителя или брокера:

1. Допустим, производитель `Producer` записывает сообщение `C` со смещением `2` в топик Kafka и не получает подтверждение записи от лидера партиции (`acks = 1`).
2. Производитель `Producer` осуществляет повторные попытки записи сообщения, пока не истекут значения параметров retries и delivery.timeout.ms.
3. При успешной записи производитель получает подтверждение от брокера.

В данной семье дублирование сообщений `C` может произойти, если после записи сообщения в топик случился сбой брокера или производителя. Результатом этого становится два одинаковых сообщения, записанных под разными смещениями. Если производитель записывает сообщение повторно и повторно получает записи, дублирования могут несколько сообщений за одну операцию записи.

Описание семантики и возможной проблемы на стороне потребителя:

1. Потребитель `Consumer 1` читает записанные сообщения `A` и `B` из Kafka.
2. Предположим, что случился сбой потребителя `Consumer 1` после обработки `A` и до обработки `B`, и он находится в одном пакете. Погда в зависимости от логики приложения потребитель может смещение `A` может быть не выполнен.
3. Так как коммит смещений `A` и `B` не выполнен, чтение или же его требований может новый потребитель `Consumer 2`.
4. При успешной обработке сообщения `Consumer 2` выполняет коммит смещений `A` и `B`.

В данной семье дублирование обработки сообщений `A` может произойти, если после обработки случился сбой потребителя и не удалось выполнить коммит смещения.

Exactly once

Семантика **exactly once** предполагает отсутствие случаев дублирования сообщений, встречающихся в семантике доставки at least once.

Для семантики exactly once обычно применяются следующие механизмы:

- Идемпотентный производитель.
- Транзакции.

Ниже описаны механизмы и их применение в семантике exactly once.

Идемпотентный производитель

Идемпотентный производитель – производитель, каждому из которых назначается идентификатор производителя (**PID**). PID включается каждый раз, когда данный производитель отправляет сообщения брокеру Kafka.

Каждое сообщение от идемпотентного производителя получает последовательно увеличивающийся порядковый номер (**sequence**), отличный от смещения.

Для каждой партиции топика, которой производитель отправляет сообщения, поддерживается отдельная последовательность. Брокер для каждой партиции отслеживает комбинацию (последовательно увеличивающийся) **PID+порядковый номер**. Если полученное сообщение с меньшим порядковым номером, чем предыдущее записанное, оно игнорируется.

Таким образом сообщение будет записано только один раз независимо от того, сколько попыток отправки было предпринято.

Идемпотентный производитель включается при помощи присвоения параметру `enable.idempotence` значения `true`.

Ниже описаны значения параметров, которые должны быть установлены, чтобы функция идемпотентного производителя работала без ошибок:

- max.in.flight.requests.per.connection — не более `5`;
- retries > `0`;
- acks = `all`.

ПРИМЕЧАНИЕ

- Идемпотентный производитель может быть использован в качестве компонента гибридного варианта семантики at least once.
- Функция идемпотентного производителя может быть сохранена, только если повторная запись сообщения была выполнена при помощи внутренних настроек производителя, как описано в семантике at least once, а не внешнего приложения.

Транзакции

Транзакция – атомарная запись в один или несколько топиков и партиций Kafka. Все сообщения, включенные в транзакцию, будут успешно записаны, либо ни одно из них не будет записано. Если транзакция прервалась, все сообщения транзакции недоступны для чтения потребителем.

Работа транзакций в Kafka основана на следующих компонентах:

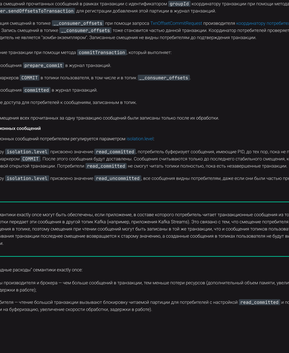
- **Идентификатор транзакции** (transactional.id) — индивидуальный идентификатор, присваиваемый производителю транзакций.
- **Координатор транзакций (transaction coordinator)** — модуль, работающий внутри каждого брокера Kafka. Только один координатор владеет зарегистрированным `transactional.id`. Для нормальной работы координатора используется четыре внутренних топика Kafka и процесс выбора лидера. Координатор транзакций – единственный компонент, который может читать и записывать журнал транзакций.
- **Журнал транзакций** (`__transaction_state`) – внутренний топик Kafka, который хранит последнее состояние транзакции. Возможные состояния транзакции:
  - `ongoing`
  - `prepare_commit`
  - `committed`

  Каждый координатор транзакций управляет некоторыми партициями в журнале транзакций, является лидером для них.
- **Эпоха** (epoch) – часть метаданных, хранящая количество подключений каждого идентификатора транзакции.
- **Зомби-экземпляр** — экземпляр производителя, который начал запись сообщений и потерял недолго, не завершив запись.

Настройка и работа в рамках транзакции производителя при помощи API класса KafkaProducer и другие API, связанных с работой транзакций.

Семантика exactly once

Общий принцип семантики exactly once в Kafka показан на рисунке ниже.



Exactly once

Последовательность записи транзакций

1. Поиск координатора транзакций при помощи запроса FindCoordinatorRequest. Производитель подключается к любому известному брокеру кластера, чтобы узнать местоположение его координатора.
2. Регистрация производителя с координатора транзакции при помощи синхронного запроса InitPidRequest. При этом производитель указывает свой идентификатор (`transactional.id`).

   Данный запрос включает:
   - Возвращение одного и того же PID для одного `transactional.id` будущим экземпляром производителя.
   - Увеличение и возвращение этого же PID при попытке предыдущий "зомби экземпляр" производителя блокируется и не может продолжить свою транзакцию.
   - Удаление любой транзакции, оставшейся не завершенной предыдущим экземпляром производителя.
   - Создание записи (`init`) в журнале `__transaction_state` для выданного `transactional.id`, если это первая транзакция для производителя.
3. Запуск транзакции при помощи вызова метода `producer.beginTransaction`. После запуска транзакции все записанные сообщения будут частью транзакции. В журнале `__transaction_state` для этого `transactional.id` меняется состояние транзакции на `ongoing`.
4. Цикл **чтение – преобразование – запись** сообщений:
   - Регистрация новой партиции в рамках транзакции при помощи запроса AddPartitionsToTxnRequest.
   - Запись сообщений (включая PID, эпоху и порядковый номер) в партицию назначения при помощи метода `producer.send`.
   - Отправка смещений прочитанных сообщений в рамках транзакции с идентификатором `groupId` координатору транзакции при помощи метода `producer.sendOffsetsToTransaction` для контроля добавления этой партиции в журнал транзакций.
   - Регистрация смещений в топике `__consumer_offsets` при помощи запроса TxnOffsetCommitRequest производителем координатору потребителей (включая PID и эпоху). Запись смещений в топике `__consumer_offsets` тем отличается частой данной транзакции. Координатор потребителей проверит, что производитель им является "зомби экземпляром". Записанные смещения не видны потребителям до подтверждения транзакции.
5. Подтверждение транзакции при помощи метода `commitTransaction`, который выполняет:
   - Запись сообщения `prepare_commit` в журнал транзакций.
   - Запись маркеров `COMMIT` в топики пользователя, в том числе и в топик `__consumer_offsets`.
   - Запись сообщения `committed` в журнал транзакций.
   - Открытие доступа для потребителей к сообщениям, записанным в топик.

Таким образом, смещения всех прочитанных за одну транзакцию сообщений были записаны только после их обработки.

Чтение транзакционных сообщений

Чтение транзакционных сообщений потребителями регулируется параметром isolation.level.

- Если параметру `isolation.level` присвоено значение `read_committed`, потребитель буферизует сообщения, имеющие PID до тех пор, пока не прочитает сообщение с маркером `COMMIT`. После этого сообщения будут доступны. Сообщения откладываются до последнего стабильного смещения, которое меньше смещения первой открытой транзакции. Потребитель `read_committed` не смотрит читать полностью, пока есть незавершенные транзакции.
- Если параметру `isolation.level` присвоено значение `read_uncommitted`, все сообщения видны потребителям, даже если они были частью прерванной транзакции.

Гарантии семантики exactly once могут быть обеспечены, если приложение, в составе которого потребитель читает транзакционные сообщения из топика Kafka, после обработки прочитанных сообщений в другой топик Kafka (например, производителем Kafka Streams). Это связано с тем, что чтение сообщений потребителем сохраняется в виде сообщения в топике, поэтому сообщения при чтении сообщений могут быть записаны в той же транзакции, что и сообщения топиков пользователя. В случае прерывания транзакции последние сообщения возвращаются в топики, а сохранные сообщения не будут видны потребителям.

Вероятные "накладные расходы" семантики exactly once:

- в части работы производителя и брокера — чем больше сообщений в транзакции, тем меньше потери ресурсов (дополнительной обработки памяти, увеличение скорости обработки, задержки и работы);
- в части потребителя — чтение большой транзакции вызывает всплеск использования памяти для потребителей с настройкой `read_committed` и потери ресурсов (объем памяти на буферизацию, увеличение скорости обработки, задержки и работы).

Репликация в Kafka

Ольга Семке

Содержание

Назад к содержанию

- Фактор репликации в Kafka
- Синхронизация реплик
- Рекомендации для настройки параметров репликации
  - Фактор репликации
  - Количество партиций топика
  - Синхронизированные реплики
- Дополнительные возможности
- Настройка репликации
  - Настройка кластера
  - Настройка топика

В данной статье описаны принципы репликации в Kafka и способы ее настройки.

ПРИМЕЧАНИЕ

- Общие концепции хранения и основные компоненты Kafka описаны в статье Концепции хранения в Kafka.
- Принципы и алгоритмы взаимодействия Kafka и ZooKeeper описаны в статье Выбор лидера в ZooKeeper.

Репликация — механизм, который создает и распределяет на брокерах точные копии каждой партиции топика в Kafka.

Репликация обеспечивает доступность сообщений в случае сбоев или технического обслуживания.

Фактор репликации в Kafka

Фактор репликации (**replication factor**) — параметр, определяющий количество копий (**реплик**) каждой партиции.

Каждая реплика партиции записывается на отдельный **брокер-фолловер** (**follower**). Реплик каждой партиции не может быть больше, чем брокеров.

На рисунке ниже показано, как происходит распределение реплик одной партиции на брокерах, запись сообщений производителем, а также чтение сообщений потребителем.

Реплики, распределенные на брокерах

Операции записи сообщений выполняются через **брокера-лидера партиции** (**leader**). Операции чтения сообщений могут быть выполнены как через лидера (по умолчанию), так и через фолловеров (начиная с версии Kafka 2.4).

Выбор лидера для каждой партиции производится при помощи ZooKeeper в соответствии с протоколом алгоритма консенсуса ZAB (ZooKeeper Atomic BroadCast).

Лидеры партиций равномерно распределяются между брокерами. Каждый брокер может быть лидером для одной партиции и фолловером для другой.

Фактор репликации в Kafka для кластера определяется параметром default.replication.factor. Все топики, созданные в кластере, будут создаваться с заданным для кластера фактором репликации по умолчанию.

Также он может быть установлен индивидуально для топика:

- при создании топика с использованием командной строки;
- при создании топика с использованием приложений потоковой обработки Kafka Streams (доступно начиная с версии Kafka 2.4);
- при создании удаленного топика с использованием механизма Mirror Maker 2.

Если в конфигурации топика указано значение фактора репликации больше, чем существует брокеров в кластере, топик не будет создан.

Фактор репликации для созданного топика не может быть переопределен. Если для всего кластера значение фактора репликации изменено, только новые топики будут создаваться с новым фактором репликации.

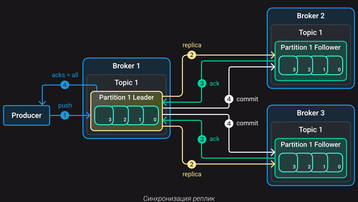
Синхронизация реплик

Синхронизация реплик актуальна при значении фактора репликации больше `1` и значении `all`, присвоенном параметру acks. Такое сочетание параметров приближает к достижению гарантий доставки сообщений exactly once.

Взаимодействие между лидером и фолловерами основано на модели обмена сообщениями, использующейся в ZooKeeper.

Журналы фолловеров стремятся быть копией журнала лидера, иметь одинаковое смещения и сообщения в одном и том же порядке. Реплика, содержащая все сообщения, записанные в журнал лидера, называется **синхронизированной репликой (In-Sync Replica, ISR)**.

Ниже на рисунке показано, как достигается синхронизация реплик.



Синхронизация реплик

Основной параметр, контролирующий синхронизацию реплик — replica.lag.time.max.ms. Это временной период, по истечении которого лидер принимает решение, что реплика записала (если произошло подтверждение записи) или не записала (не произошло подтверждение записи) сообщение.

Критерий успешности записи сообщения в целом определяет параметр min.insync.replicas — минимальное количество реплик, которые должны подтвердить запись (также может быть установлен и изменен для каждого топика индивидуально при помощи параметра min.insync.replicas). Только при достижении числа реплик, указанного в этом параметре, лидер передает данные о записанной партиции (метаданные) и о созданных ISR в контроллер.

После получения сообщения лидер предлагает запись сообщения репликам и ждет от них подтверждения. Получив подтверждение от нужного количества реплик (параметр `min.insync.replicas`), лидер отправляет сигнал о фиксации сообщений в репликах и подтверждение для производителя. Только тогда сообщение становится доступно для чтения.

Рекомендации для настройки параметров репликации

Фактор репликации

Существует общая рекомендация устанавливать значение `3` для фактора репликации.

На самом деле точное значение параметра зависит от требований и ресурсов вашей системы. При выборе значения стоит обратить внимание на следующее:

- Высокий фактор репликации обеспечивает лучшую устойчивость системы.
- Высокий фактор репликации может вызвать более высокую задержку, с которой сталкиваются производители, поскольку данные необходимо реплицировать на все брокеры реплик, прежде чем будет возвращено подтверждение.
- Высокий фактор репликации требует большего дискового пространства из-за большого количества копий данных на дисках для обеспечения высокой доступности (High Availability, HA).

Количество партиций топика

Устанавливая фактор репликации, следует помнить, что на работу системы также влияет параметр num.partitions — количество партиций всех топиков в кластере (может быть установлен при помощи опции `--partitions` для отдельного топика). Фактор репликации и количество партиций являются параметрами, работающими в связке, и от их настройки зависит производительность и долговечность системы.

При выборе значения параметра `num.partitions` стоит обратить внимание на то, какие возможности и ограничения вызывает большое количество партиций:

- Дает возможность объединить больше потребителей в группу для масштабирования.
- Позволяет использовать более эффективно имеющиеся брокеры кластера.
- Увеличивает нагрузку на ZooKeeper — выбор лидера, поддержка списка синхронизированных реплик.
- Увеличивает количество файлов, созданных и открываемых на брокерах.

Синхронизированные реплики

Параметр `min.insync.replicas` в сочетании с параметром acks, которому присвоено значение `all`, оказывает влияние на гарантию доставки сообщений при использовании семантики exactly once.

Параметр `min.insync.replicas` рекомендуется устанавливать меньше, чем фактор репликации, иначе даже при отказе одного брокера запись сообщения не будет подтверждена из-за недостаточности синхронизированных реплик.

Стандартный рекомендуемый сценарий для обеспечения высоких гарантий надежности:

- default.replication.factor = `3`
- min.insync.replicas = `2`
- acks = `all`

Дополнительные возможности

Начиная с Apache Kafka 2.4 можно настроить потребителей для чтения из ближайших синхронизированных реплик.

Для выбора ближайших синхронизированных реплик используется плагин RackAwareReplicaSelector. Для его работы должны быть настроены параметры:

- broker.rack — указание принадлежности к определенной стойке для брокеров. Для каждого брокера должна быть назначена стойка. Для эффективной работы плагина рекомендуется настроить одинаковое количество брокеров на стойку — таким образом реплики будут равномерно распределены по брокерам.
- client.rack — настройка идентификатора стойки для подключения потребителей.

Плагин `RackAwareReplicaSelector` пытается сопоставить `client.rack` потребителя с доступными `broker.racks`. Затем он выбирает реплику, имеющую тот же идентификатор стойки, что и потребитель.

Настройка репликации

Настройка кластера

После добавления и установки сервиса ***Kafka*** в составе кластера ADS настройка фактора репликации и количества партиций выполняется на странице конфигурирования сервиса ***Kafka*** в интерфейсе ADCM. Для этого перейдите в активное состояние переключатель ***Show advanced***, раскройте узел ***server.properties*** в дереве конфигурационных настроек и введите новое значение для параметров.

Для изменения значения по умолчанию для параметра min.insync.replicas и других параметров брокера, отсутствующих в интерфейсе ADCM, используйте поле ***Add key:value***: выберите ***Add property*** и введите наименование параметра и его значение.

После изменения параметров при помощи интерфейса ADCM перезагрузите сервис ***Kafka***. Для этого примените действие Restart, нажав на иконку в столбце ***Actions***.

После ввода новых значений и параметров все изменения отображаются в конфигурационном файле */usr/lib/kafka/config/server.properties*.

Настройка топика

Настройка фактора репликации для отдельного топика выполняется в командной строке. Для этого используется скрипт *kafka-topics.sh* с применением опции `--replication-factor` при создании нового топика.

Значение параметра `min.insync.replicas` может быть изменено при помощи опции `--config`, которая переопределяет параметры конфигурации для существующего топика или устанавливает параметры конфигурации при создании нового топика.

Пример настройки фактора репликации и параметра min.insync.replicas при создании топика при помощи командной строки

```
$ /usr/lib/kafka/bin/kafka-topics.sh --create --topic new-topic --replication-factor 3 --config min.insync.replicas=2 --
bootstrap-server localhost:9092
```

ПРИМЕЧАНИЕ

Для получения информации об основных принципах работы в Kafka можно обратиться к статье Начало работы с Kafka.

Обзор KRaft

Ольга Семина

Содержание

Назад к содержанию

- Преимущества KRaft
- Кворум контроллера
- Алгоритм Raft
  - Обзор
  - Голосование
  - Репликация сообщений
  - KRaft
- Управление метаданными
  - Запись метаданных
  - Репликация метаданных
- KRaft в ADS

Kafka Raft (KRaft) – это протокол, основанный на алгоритме консенсуса Raft, при помощи которого осуществляется управление метаданными кластера Kafka. KRaft организовывает хранение информации о конфигурации кластера, обеспечивает синхронизацию брокеров Kafka при репликации данных без использования механизмов выбора лидера и хранения метаданных, предоставляемых ZooKeeper.

ПРИМЕЧАНИЕ

Возможность включения режима KRaft существует начиная с версии Kafka 3.6.2.

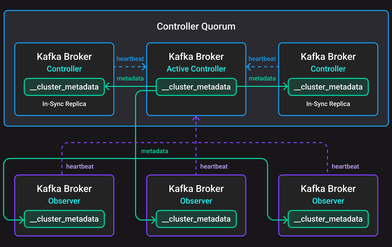
Преимущества KRaft

Главная характеристика KRaft заключается в отсутствии сервиса-посредника, что влечет за собой следующие преимущества:

- Быстрое и эффективное хранение и распространение метаданных: обновления отправляются непосредственно в кворум контроллера, устраняя дополнительный узел передачи (ZooKeeper).
- Брокеры и контроллеры могут хранить метаданные локально в своей кеш-памяти и обращаться к ним, не используя соединение с внешним сервисом.
- Уменьшение задержек при завершении работы брокеров.
- Предполагается, что протокол KRaft позволит значительно увеличить предел для количества партиций для одной ноды и целого кластера.

Кворум контроллера

Основная концепция KRaft основана на создании **кворума контроллера** – набора специализированных брокеров, которые отвечают за хранение и своевременное обновление метаданных кластера.



Кворум контроллера в режиме KRaft

Для назначения контроллеров в Kafka используется параметр process.roles со значением `controller`.

Кворум контроллера обслуживает только внутренний топик `__cluster_metadata` с метаданными Kafka. Топик содержит одну партицию, в которую записывается вся информация о текущем состоянии брокеров Kafka, а также топиков и партиций. Лидер партиции — **активный контроллер** — выбирается на основе алгоритма консенсуса Raft путем голосований за **эпоху лидера**. Активный контроллер кворума отвечает за запись изменений метаданных в топик `__cluster_metadata`.

Контроллеры, не являющиеся лидером, хранят точную копию топика с метаданными как in-sync replicas (ISR) и имеют право голосовать за новую эпоху лидера. В результате выборов каждый из этих избирателей может стать лидером.

Брокеры, у которых параметр process.roles имеет значение `broker`, не могут голосовать за эпоху лидера, они являются **наблюдателями** (**observer**). Наблюдатели работают как обычные брокеры Kafka, но при этом хранят копию топика `__cluster_metadata`, перезаписывая каждый раз в собственную кеш-память состояние метаданных кластера при изменении топика.

Также активный контроллер осуществляет отслеживание состояния всех контроллеров и брокеров при помощи тайм-аутов и heartbeat-сообщений, которые посылают брокеры. При остановке брокера лидер перераспределяет партиции на другие брокеры и обновляет информацию о них в топике метаданных.

Для согласования реплик используется маркер **high water mark (HWM)** – наибольшее смещение в логе, которое было записано во все ISR. Последователи кешируют его значение и обновляют при каждом изменении лога лидера, таким образом поддерживая согласованное состояние кластера. При смене лидера все партиции последователей усекают свой журнал до HWM, зафиксированного новым лидером, и все сообщения выше HWM становятся нечитаемыми. После усечения журнала последователь может продолжить репликацию любых новых сообщений от лидера. Как только все новые сообщения реплицируются на большинство узлов, HWM увеличивается.

Кроме параметра `process.roles`, для брокеров кластера Kafka, настраиваемого для работы в режиме KRaft, должны быть определены параметры:

- controller.quorum.voters.roles — адреса узлов, участвующих в кворуме контроллера.
- node.id — идентификатор узла, который должен быть уникальным целым числом и назначаться во время установки/расширения. Не может совпадать с идентификатором любого другого брокера или контроллера в кластере.
- controller.listener.names — разделенный запятыми список имен слушателей, используемых контроллером. При общении с кворумом контроллера брокер в режиме KRaft всегда будет использовать первого слушателя в этом списке.
- metadata.log.dir — директория для хранения метаданных кластера. Имя директории не должно совпадать с любым каталогом брокера, по умолчанию */kafka-meta*.

Алгоритм Raft

Обзор

Алгоритмы консенсуса Raft и ZAB, использующийся в ZooKeeper, имеют следующие концептуальные сходства:

- единственный лидер, отвечающий за обработку данных, в любой момент времени;
- отдельные процессы для выборов лидера и репликации журналов;
- общая нацеленность на обеспечение согласованности данных в распределенных системах.

Архитектура Raft имеет более простую реализацию, в которой выделены следующие роли для работы узлов:

- **Лидер** кластера, который обрабатывает запросы клиента и контролирует репликацию данных.
- **Кандидат**, пытающийся стать лидером.
- **Последователь**, записывающий сообщения и обрабатывающий запросы от лидера и кандидатов.

Голосование

Лидер выбирается случайным образом при первом запуске, а остальные узлы начинают свою работу в качестве последователей.

Лидер отслеживает работоспособность последователей при помощи тайм-аутов и периодических heartbeat-сообщений, а также отправляет свои heartbeat-сообщения вместе с сообщениями топиков (если они есть) или без них.

Если последователь в течение тайм-аута не получает heartbeat-сообщений от лидера, он становится кандидатом в лидеры.

Каждый кандидат отправляет остальным узлам запрос на лидерство и принимает такие же запросы от остальных. Кандидат, получивший большинство голосов, становится лидером. Лидер выбирается на период, так называемый **срок**. Срок идентифицируется как **эпоха** – число, которое увеличивается монотонно с каждым новым лидером.

Репликация сообщений

Лидер отправляет всем последователям запрос на добавление новой записи и считает транзакцию успешной только после получения от большинства узлов подтверждения записи сообщения.

Сообщения всегда добавляются последовательно.

Если внешний клиент подключается к кластеру через последователя, запрос все равно перенаправляется лидеру.

KRaft

Дополнительно в протоколе KRaft появляется способ хранения метаданных, основанный на протоколе репликации логов Kafka, где и используется кворум контроллера. Основной принцип заключается в том, что каждый брокер кластера стремится хранить реплику топика метаданных и не требует соединения с ZooKeeper для получения состояния кластера (лидеры партиций топиков, новые брокеры и т.д.) и, соответственно, задержек и значительного объема памяти, так как процесс хранения метаданных как лога является частью работы алгоритмов Kafka.

Управление метаданными

Запись метаданных

Каждый новый брокер или контроллер, который присоединяется к кластеру, запрашивает копию внутреннего топика __cluster_metadata и создает его реплику, одновременно кешируя данные топика, т.е. сохраняет данные о полном состоянии кластера.

Если брокер запускается после сбоя, он считывает самые последние изменения, доступные в его кеше метаданных. После этого он запрашивает у активного контроллера события, произошедшие с момента последнего обновления в `__cluster_metadata`, и в ответ получает последние изменения в виде снепшотов состояния кластера.

Репликация метаданных

Для Kafka, работающей в режиме KRaft, HWM отслеживается и учитывается лидером для общего количества реплик топика метаданных (не только для реплик ISR, как в обычной Kafka).

HWM не может быть увеличена, пока все сообщения, записанные лидером, не будут сохранены большинством реплик. Если произошла смена лидера, эпоха увеличивается. Для согласования своих логов с лидером последователи отправляют запрос, содержащий последнее смещение и последнюю эпоху, во время которой были записаны сообщения.

На основании таких запросов лидер отслеживает работоспособность брокеров и в ответ отправляет информацию о новой эпохе и последнем смещении. Далее реплики догоняют лидера или отсекают свой лог до HWM, переданного лидером. После того, как большинство брокеров подтверждает запись последнего смещения, HWM увеличивается. После усечения журнала ведомый может продолжить репликацию любых новых сообщений от лидера.

KRaft в ADS

Начиная с версии ADS 3.6.2.2.b1 доступна установка сервиса Kafka в режиме KRaft с использованием интерфейса ADCM. Данный способ автоматически устанавливает для брокеров необходимые параметры.

ВНИМАНИЕ

- Возможность использования протокола KRaft для Kafka предоставляется в режиме технической предварительной версии и не рекомендуется для использования в продуктовой среде. Arenadata рекомендует вам изучить эту функцию в непроизводственных средах и при необходимости оставить отзыв о своем опыте.
- Для кластеров, в которых сервис Kafka используется вместе с ZooKeeper, может быть выполнена миграция в режим KRaft.
- Возможность миграции из KRaft в ZooKeeper отсутствует.
- Для кластера ADS, в котором планируется установка сервиса Kafka в режиме KRaft, установка сервиса ZooKeeper не требуется.

Обзор Kafka Tiered Storage

Ольга Семме

Содержание

Назад к содержанию

Преимущества использования Tiered Storage

Архитектура Tiered Storage

Ограничения Tiered Storage

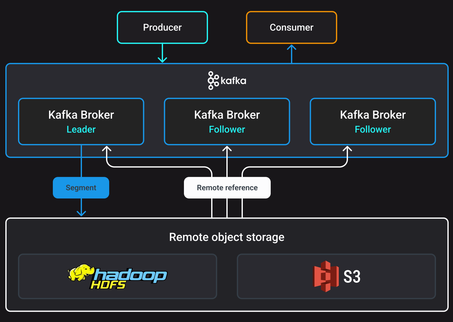
Kafka Tiered Storage в ADS

Настройка Tiered Storage

Kafka Tiered Storage — это опция, которая запускает многоуровневый подход к хранению данных в кластере Kafka, доступная начиная с версии Kafka 3.6.0.

После активации опции в Kafka для хранения устанавливается два уровня:

- **Локальный уровень** — локальные диски брокеров Kafka, хранящие сегменты лога аналогично обычным брокерам Kafka.
- **Удаленный уровень** — системы хранения файлов (например, HDFS или S3), на которых хранятся закрытые сегменты лога.



Kafka Tiered Storage

Лидер партиции, используя потоки, управляемые при помощи специальных компонентов Remote Manager, копирует все закрытые сегменты в хранилище (период, по истечении которого закрытые сегменты отправляются во внешнее хранилище, определяется при помощи параметра брокера remote.log.manager.task.interval.ms). Далее лидер сохраняет и публикует ссылку, содержащую индексы и метаданные **remote-сегмента** — закрытого сегмента, сохраненного в хранилище (данные эпохи лидера, производителя, смещение и данные о хранении сегмента в хранилище). Закрытые сегменты удаляются с брокера по истечении времени локального хранения (local.retention.ms) или при достижении максимального размера локального хранения (local.retention.bytes).

Реплики, стремясь стать ISR, копируют данные, записанные на лидер, при этом кешируют ссылки к remote-сегментам.

Запись и удаление сообщений являются идемпотентными процессами и выполняются в виде транзакций.

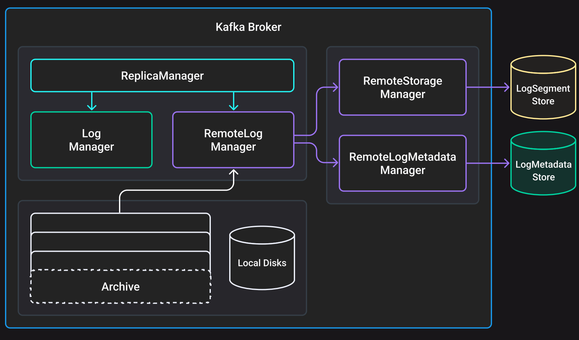
Взаимодействие производителей и потребителей с брокерами Kafka не изменяется после включения опции Tiered Storage. Запись сообщений производится в активные сегменты, расположенные на локальном уровне, а для каждого запроса потребителя на чтение сообщения из удаленного уровня создается отдельный поток, который обслуживается при помощи Remote Manager компонентов, как и потоки для записи сегментов в хранилище.

Преимущества использования Tiered Storage

- Возможность долговременного хранения данных во внешних хранилищах.
- Уменьшение объема данных на брокерах Kafka, и, следовательно, объема данных, которые необходимо копировать во время восстановления и ребалансировки.
- Упрощение эксплуатации больших кластеров Kafka с длительным хранением данных.

Архитектура Tiered Storage

После активации опции Tiered Storage на брокере Kafka запускаются компоненты, отвечающие за управление remote-сегментами.



Архитектура Kafka Tiered Storage

RemoteLogManager — внутренний компонент, запускающийся на каждом брокере Kafka. Через него брокер взаимодействует с remote-сегментами. RemoteLogManager не имеет общедоступного API.

Постоянный компонент ReplicaManager, управляющий репликами партиций, вызывает RemoteLogManager для управления сегментами.

RemoteLogManager, в свою очередь, передает команду о копировании сегментов в хранилище или удалении их компоненту RemoteStorageManager и поддерживает соответствующие метаданные remote-сегмента для RemoteLogMetadataManager.

RemoteStorageManager — интерфейс, обеспечивающий жизненный цикл remote-сегментов журнала и индексов.

Remote Log Metadata Manager — интерфейс, обеспечивающий жизненный цикл метаданных remote-сегмента журнала со строго согласованной семантикой. Существует реализация по умолчанию, использующая внутренний топик `__remote_log_metadata`. Пользователи могут подключить свою собственную реализацию, если они намерены использовать другую систему для хранения метаданных remote-сегмента.

Ограничения Tiered Storage

- После включения для отдельного топика опцию нельзя отключить.
- Опцию невозможно использовать для топиков, в которых для политики очищения установлено значение compact.
- Для существующего топика невозможно перенести сегменты обратно на локальные брокеры, увеличив время локального хранения данных.
- Нельзя использовать с дисковым массивом (JBOD) в качестве локального брокера.

Kafka Tiered Storage в ADS

Опция Tiered Storage в ADS основана на реализации Aiven, которая к встроенному в Kafka RemoteStorageManager добавляет следующие возможности:

- Сжатие сегментов при помощи механизма **chunking** — деление оригинальных сегментов на части и сжатие их перед отправлением в хранилище.
- Шифрование сегментов с поддержкой ротации ключей.
- Оптимизация числа обращений к удаленному хранилищу.
- Абстракция удаленного хранилища в виде StorageBackend — контейнера байтовых массивов на основе сообщений ключ/значение.

Дополнительно в ADS реализована поддержка HDFS в качестве удаленного хранилища, для которой на основе решения Aiven разработан StorageBackend, обеспечивающий:

- Поддержку конфигурационных XML-файлов.
- Поддержку аутентификации при помощи Kerberos.
- Экспорт метрик клиента HDFS.

Настройка Tiered Storage

Настройка работы опции Tiered Storage выполняется в конфигурационном файле */etc/kafka/conf/server.properties* и состоит из нескольких блоков параметров:

- Включение опции Tiered Storage при помощи параметров брокера remote.log.storage.system.enable или топика remote.storage.enable.
- Настройка компонентов Remote Manager.
- Настройка серверной части хранилища (storage backend).
- Настройка кеша фрагментов выборки (chunking) — если это поддерживается в storage backend.

ПРИМЕЧАНИЕ

Настройка Tiered Storage в ADS при помощи интерфейса ADCM описана в статье Настройка и использование Tiered Storage.

Обзор kTLS

Ольга Семме

Содержание

Назад к содержанию

Kafka и TLS

Kafka и kTLS

Kernel TLS (kTLS) — реализация протокола TLS, основанная на переносе части операций, в том числе шифрования и дешифрования, на уровень ядра Linux, что уменьшает нагрузку на процессор и улучшает производительность.

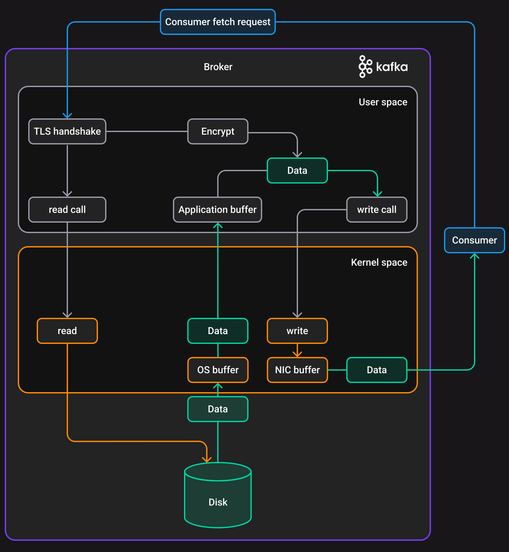
В Kafka поддержка kTLS осуществляется начиная с версии ADS 3.9.0.1 b1.

Ниже описано, как осуществляется взаимодействие потребителя и брокера Kafka при использовании традиционного протокола TLS и при использовании протокола kTLS.

Основные понятия, используемые для описания:

- **Transport layer security (TLS)** — криптографический протокол, обеспечивающий безопасную передачу данных по сети при обмене данными между двумя приложениями. Является усовершенствованной версией протокола secure sockets layer (SSL).
- **Пространство ядра** (**kernel space**) — область памяти, зарезервированная для ядра операционной системы, его расширений и драйверов устройств.
- **Пользовательское пространство** (**user space**) — адресное пространство виртуальной памяти операционной системы, предназначенное для пользовательских программ, отделенное от пространства ядра для обеспечения безопасности и стабильности системы.
- **TLS-рукопожатие** (**TLS handshake**) — взаимодействие клиента и сервера при подключении. При рукопожатии происходит определение используемой версии TLS, используемых шифров, обмен ключами и проверка подписей центра сертификации SSL, генерация сеансовых ключей для использования симметричного шифрования.
- **Сетевая интерфейсная карта** (**Network Interface Controller, NIC**) — аппаратное устройство для взаимодействия с сетью.
- **Буфер сетевой карты** (**NIC buffer**) — область памяти, используемая сетевой картой для временного хранения данных при приеме и передаче.
- **Сокет** (**socket**) — абстракция в Linux для обмена данными между процессами или устройствами. В ядре Linux сокеты реализованы как структуры данных, позволяющие системе обрабатывать сетевые операции, не обращаясь напрямую к аппаратным деталям.
- **Файловый дескриптор** — натуральное число (идентификатор), закрепленное за определенным потоком ввода-вывода.

Kafka и TLS



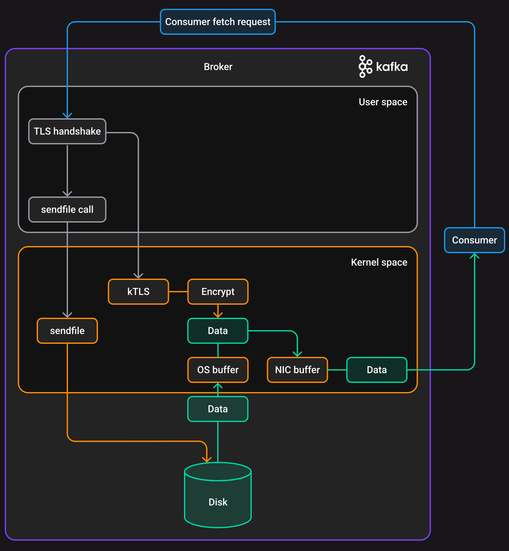
Kafka и традиционный TLS

При установке соединения клиента (потребителя) и брокера Kafka после осуществления TLS-рукопожатия приложение в пользовательском пространстве производит системный вызов `read`. При этом в ядро передается файловый дескриптор, указатель на буфер приложения и другие данные для чтения. Данные копируются через буфер ядра в буфер приложения, шифруются (в пользовательском пространстве) и, используя системный вызов `write`, записываются в буфер сетевой карты (в пространстве ядра), передавая дескриптор файла, указатель на буфер данных и количество байтов для записи, и отправляются потребителю.

Недостатки работы Kafka с традиционным TLS:

- Каждый пакет данных копируется дважды между ядром и пользовательским пространством, что вызывает задержки из-за дополнительных операций и переключений между ядром и пользовательским пространством.
- Высокая нагрузка на процессор из-за выполнения шифрования в программном режиме.

Kafka и kTLS



Kafka и kTLS

При установке соединения клиента (потребителя) и брокера Kafka после осуществления TLS-рукопожатия (рукопожатие по-прежнему выполняется в пользовательском пространстве) приложение производит системный вызов `sendfile`, который копирует данные между двумя файловыми дескрипторами, то есть данные копируются непосредственно из кеша ядра в сетевой буфер.

Для осуществления шифрования данных в пространстве ядра создается специальный файловый дескриптор сокета TLS (KTLS), куда извлекается информация о сеансовых ключах, идентификаторах и шифровании данных из библиотеки TLS. Данные шифруются непосредственно в ядре перед отправкой в сеть.

Зашифрованные данные расшифровываются в ядре перед передачей в пользовательское пространство, если это предусмотрено приложением.

Преимущества:

- Обеспечение принципа **нулевого копирования** (**zero-copy**) — системный вызов `sendfile` копирует данные между одним дескриптором файла и другим сразу внутри ядра вместо комбинации вызовов `read` и `write`, которые выполняют передачу данных в пользовательское пространство и чтение из него. Это снижает нагрузку на процессор и повышает производительность и эффективность, что имеет особенное значение при передаче большого объема данных.
- Снижение нагрузки на процессор за счет переноса операций шифрования/дешифрования в ядро.
- Улучшенная пропускная способность и уменьшенная задержка при передаче данных по защищенным каналам.
- Прозрачная интеграция с существующими TLS-библиотеками (например, OpenSSL) через API ядра.

ВАЖНО

Для поддержки kTLS требуется ядро операционной системы версии 5.9 и выше. По умолчанию режим kTLS включен. Но если версия ядра ниже требуемой, то произойдет откат в обычный режим.

Обзор Cruise Control

Ольга Самоед

Содержание

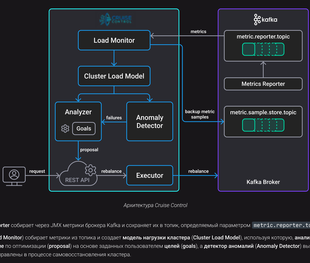
Назад к содержанию

- Архитектура
- Load Monitor
- Analyzer
- Anomaly Detector
- Executor

Архитектура

Cruise Control — инструмент, предназначенный для управления высоконагруженными кластерами Kafka или кластерами с большим количеством хостов, позволяющий оценивать состояние брокеров, а также выполнять балансировку нагрузки.

Работа с Cruise Control в ADS доступна начиная с 3.9.0.1.81.



Архитектура Cruise Control

Интерфейс **Metrics Reporter** собирает через JMX метрики брокера Kafka и сохраняет их в топик, определяемый параметром `metric.reporter.topic`.

Монитор нагрузки (**Load Monitor**) собирает метрики из топика и создает **модель нагрузки кластера** (**Cluster Load Model**), используя которую, **анализатор** (**Analyzer**) генерирует **предложения** (**proposal**) по оптимизации. Основываясь на основе заданных пользователем **целей** (**goals**), и **детектор аномалий** (**Anomaly Detector**) выявляет аномалии, которые могут быть исправлены в кластере Kafka (определенные параметрами).

Модель нагрузки кластера – программная модель, отражающая текущее распределение реплик кластера с гранулированностью данных о нагрузке репликами для различных ресурсов.

Пользователь может получить сгенерированное предложение, сделав запрос `GET` на REST API Cruise Control, и при помощи запроса `POST` запустить **перебалансировку** (**rebalance**) кластера Kafka. Перед запуском перебалансировки пользователь может отредактировать цели и/или настроить самовосстановление кластера при необходимости.

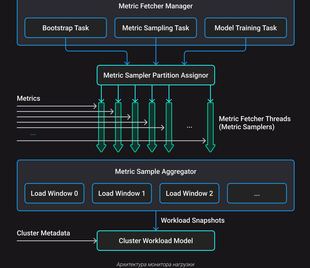
Исполнитель (**Executor**) отвечает за выполнение предложений по оптимизации, полученных от анализатора.

Обработанные монитором нагрузки метрики также передаются в подключаемое хранилище, в котором сохраняются для восстановления кластера. В ADS это специальные топики в кластере Kafka (определяемые параметрами `*.metric.sample.store.topic`). Когда Cruise Control будет перезапущен, он загрузит метрики из этого хранилища для заполнения монитора нагрузки.

Ниже подробно описаны компоненты Cruise Control.

Load Monitor

Для создания модели нагрузки кластера внутри монитора нагрузки используется актуальная **выборка метрик** (**metrics sampler**) в сочетании с метаданными кластера.



Архитектура монитора нагрузки

Метрики по всем партициям кластера Kafka назначаются специальным потокам **Metric Fetcher Threads** при помощи **Partition Assignor**, который обеспечивает:

- одинаковое количество метрик партиций для всех потоков;
- назначение метрик партиций одного топика одному потоку.

Потоки являются частью реализации **задач выборки** (**sampling tasks**).

Существуют три типа задач-выборок:

- Metric Sampling Task — задача-выборки метрик.
- Bootstrap Task — задача-выборки bootstrap-метрик.
- Linear Model Training Task — задача обучения линейной модели.

Задачи координируются при помощи модуля **Metric Fetcher Manager**. Он создает определенное количество потоков для выполнения каждой задачи.

Все выборки метрик организовываются при помощи модуля **Metric Sample Aggregator**. Каждая выборка метрик помещается в **окно нагрузки** (**load window**) в соответствии с временной меткой выборки.

Окно нагрузки – это агрегированная информация о метриках за заданный временной интервал (временное окно). Такой интервал (окно) определяется параметром `partition.metrics.window.ms` `broker.metrics.window.ms` – для метрик брокеров Kafka).

Количество поддерживаемых окон определяется параметром `num.partition.metrics.windows` `num.broker.metrics.window` – для метрик брокеров Kafka).

Например, если временное окно составляет 1 час, а пользователь настроил хранение 168 окон Cruise Control будет хранить последнюю информацию о нагрузке за 7 дней.

Количество выборок в одном окне нагрузки определяется параметром `metric.sampling.interval.ms`. Например, если пользователь указывает временное окно длительностью 1 час, а интервал выборки – 5 минут, каждое окно должно содержать 12 выборок метрик. При помощи параметра `min.samples.per.partition.metrics.window` пользователь может определить минимальное количество выборок в каждом окне, при котором эти выборки будут включены в модель нагрузки кластера.

При запросе пользователя **Metric Sample Aggregator** возвращает все агрегированные данные окна нагрузки — **снимок нагрузки** (**workload snapshot**) в монитор нагрузки в составе модели нагрузки кластера.

Analyzer

Анализатор генерирует предложения на основе заданных пользователем целей оптимизации и модели нагрузки кластера, полученной из монитора нагрузки.

Предложение — план оптимизации для перебалансировки кластера Kafka. В предложении описывается ряд действий, целью которых является улучшение использования ресурсов кластера и балансировка нагрузки между брокерами, например, перераспределение партиций и выборы лидера.

Cruise Control позволяет задавать **жесткие** и **мягкие** цели.

Жесткая цель – цель, которая должна быть достигнута. Такие цели должны быть помечены в списке `hard.goals`. Мягкая цель – цель, которая может быть не достигнута, если это позволяет достичь всех жестких целей.

Существует набор целей, которые пользователь может добавлять и удалять из списка `goals` для получения ожидаемого результата балансировки. Пользователи могут создавать свои собственные цели и добавлять их в Cruise Control.

Ниже приведено описание всех существующих целей.

Описание целей

Цель	Описание
BrokerSetAwareGoal	BrokerSet — подмножество брокеров в кластере. Эта цель ограничивает перемещения реплик в пределах BrokerSet
RackAwareGoal	Цель, гарантирующая, что все реплики каждой партиции назначаются с учетом стойки
RackAwareDistributionGoal	Цель позволяет размещать несколько реплик партиции в одной стойке при условии идеально равномерного распределения реплик каждой партиции по стойкам
ReplicaCapacityGoal	При установке этой цели система будет пытаться сделать так, чтобы количество реплик всех брокеров в кластере было меньше заданного
DiskCapacityGoal	Цель, гарантирующая, что использование дисковых ресурсов будет ниже заданного порогового значения. Эта цель будет достигнута, если параметр `rebalance_disk` в запросе на балансировку установлен в значение `true`
NetworkInboundCapacityGoal	Цель, гарантирующая, что объем используемой входящей сети каждым брокером меньше установленного порога. Объем задается с помощью параметра `network.inbound.capacity.threshold` в *cruisecontrol.properties*
NetworkOutboundCapacityGoal	Цель, гарантирующая, что объем используемой исходящей сети каждым брокером меньше установленного порога. Объем задается с помощью параметра `network.outbound.capacity.threshold` в *cruisecontrol.properties*
CpuCapacityGoal	Цель, гарантирующая, что загрузка CPU на каждом брокере меньше установленного порога; задается с помощью параметра `cpu.capacity.threshold` в *cruisecontrol.properties*
ReplicaDistributionGoal	При задании этой цели система будет пытаться сделать так, чтобы все брокеры в кластере имели одинаковое количество реплик
PotentialNwOutGoal	Цель, гарантирующая, что потенциальная выходная мощность сети (когда все реплики становятся лидерами) на каждом брокере не превысит исходящую пропускную способность сети брокера
DiskUsageDistributionGoal	При задании этой цели система будет пытаться сделать так, чтобы разброс использования между всеми дисками в пределах одного брокера находился в определенном диапазоне. Эта цель будет достигнута, если параметр `rebalance_disk` в запросе на балансировку установлен в значение `true`
NetworkInboundUsageDistributionGoal	При задании этой цели система будет пытаться сделать так, чтобы распределение входящего сетевого трафика между всеми брокерами было равномерным
NetworkOutboundUsageDistributionGoal	При задании этой цели система будет пытаться сделать так, чтобы распределение исходящего сетевого трафика между всеми брокерами было равномерным
CpuUsageDistributionGoal	При задании этой цели система будет пытаться сделать так, чтобы дисперсия утилизации CPU между всеми брокерами была равномерной
TopicReplicaDistributionGoal	Цель, гарантирующая равномерное распределение реплик одного топика по всему кластеру
LeaderReplicaDistributionGoal	Цель, гарантирующая равномерное распределение реплик лидеров на всех брокерах кластера
LeaderBytesInDistributionGoal	При задании этой цели система будет пытаться сделать так, чтобы входящий сетевой трафик на лидеров партиций в кластере был равномерным
KafkaAssignerDiskUsageDistributionGoal	Цель, гарантирующая назначение всех реплик каждой партиции с учетом стойки
KafkaAssignerEvenRackAwareGoal	При задании этой цели система будет пытаться обеспечить одинаковое количество реплик на всех брокерах в кластере
PreferredLeaderElectionGoal	При задании этой цели система будет пытаться сделать первую реплику в списке реплик ведущей репликой партиции для всех партиций топика

Anomaly Detector

Детектор аномалий выявляет различные аномалии и после включения самовосстановления (при помощи параметра `self.healing.enabled`) инициирует попытку автоматического исправления определенных типов сбоев. Цели, используемые для самовосстановления, указываются в списке `self.healing.goals`.

Ниже приведено описание аномалий.

Описание аномалий

Аномалия	Описание
Broker failures	Сбой неглупого брокера или вывод из кластера. Это приводит к появлению офлайн-репликаций/недореплицированных партиций. В этом случае Cruise Control отправляет уведомление, и, если включено самовосстановление для данного типа аномалии, Cruise Control запускает операцию по перемещению всех офлайн-реплик на другие работоспособные брокеры в кластере. Поскольку это может происходить из-за кратких обычных отказов кластера, детектор аномалий предусматривает настраиваемый льготный период, прежде чем сработает уведомление и кластер будет восстановлен
Goal violations	Нарушается цель оптимизации. В этом случае Cruise Control отправит уведомление, и если включено самовосстановление для данного типа аномалии, Cruise Control запланированно попытается устранить нарушение цели, автоматически анализируя нагрузку и выполняя предложение по оптимизации
Disk failure	Вывод из строя одного из неглупых дисков. Такая аномалия связана только с брокером Kafka, работающим на диске JBOD. В этом случае Cruise Control отправит уведомление, и если включено самовосстановление для данного типа аномалии, Cruise Control запустит операцию по перемещению всех автономных реплик на другие работоспособные брокеры в кластере
Metric anomaly	Одна из метрик, собираемых Cruise Control, обнаруживает аномально (например, резкое увеличение метрики времени очистки журнала). При этом Cruise Control отправляет уведомление. В настоящее время не существует стандартизированной операции самовосстановления для аномалий метрик, поскольку для разных аномалий метрик требуются разные методы устранения. Пользователь может определить собственную аномалию и операцию устранения, реализовав собственные MetricAnomaly и MetricAnomalyFinder
Topic anomaly	Один или несколько топиков в кластере нарушают заданные пользователем свойства (например, некоторые партиции на диске слишком велики). В этом случае Cruise Control отправит уведомление. В настоящее время не существует стандартизированной операции самовосстановления для аномалий в топиках, поскольку для разных аномалий в топиках требуются разные методы исправления. Пользователь может определить собственную аномалию и операцию исправления, реализовав собственные методы TopicAnomaly и TopicAnomalyFinder

Executor

Исполнитель отвечает за выполнение предложений по оптимизации, полученных от анализатора. Работа исполнителя может быть прервана во время выполнения предложений. Исполнитель гарантирует, что выполнение учитывает ресурсы и не перегружает ни одного брокера. Некоторые параметры могут контролировать перемещения партиций при перебалансировке, например, `num.concurrent.partition.movements.per.broker` или `max.num.cluster.partition.movements`.

ПРИМЕЧАНИЕ

Для получения информации об установке, настройке и использовании Cruise Control в ADS обратитесь к статье Пример использования Cruise Control.

Обзор Kafka Connect

Автор: ADS

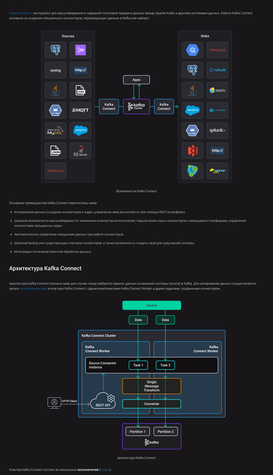
Содержание

Назад к содержанию

- Возможности Kafka Connect
- Архитектура Kafka Connect
  - Worker
  - Connector
  - Task
  - Converter
  - Transform
  - Обработка ошибок
- Kafka Connect в ADS
  - Подключение
  - Kafka
  - Schema Registry
  - ksqlDB
  - Настройка

Возможности Kafka Connect

Kafka Connect - инструмент для масштабируемой и надежной потоковой передачи данных между кластером Apache Kafka и другими системами данных. Platform Kafka Connect позволяет не создавать специальных коннекторов. Конфигурация для Kafka как кластер:



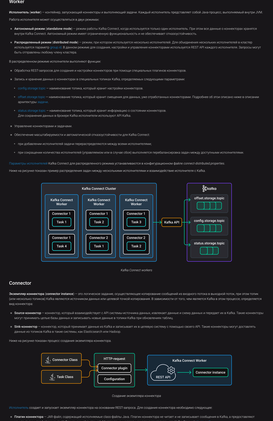
Возможности Kafka Connect

Основные преимущества Kafka Connect перечислены ниже:

- Потоковая данных для стандартных источников и целей, управление ими включается при помощи REST-интерфейса.
- Широкие возможности конфигурации и масштабируемости: изменение количества потоков, поддержание кластера, конфигурация с использованием конфигурации, управление ими.
- Автоматическое управление смещениями данных при работе коннектора.
- Широкие выбор для создания пользовательских коннекторов, а также возможность создать свой для нужных систем.
- Интеграция готовыми инструментами обработки данных.

Архитектура Kafka Connect

Архитектура Kafka Connect состоит для того, когда при передаче данных из внешней системы (source) в Kafka или для передачи данных из кластера Kafka во внешнюю систему (sink) с данным коннекторами Kafka Connect API и кластера Kafka Connect. Данные коннекторами Kafka Connect API и кластера Kafka Connect. создаются коннекторы.



Архитектура Kafka Connect

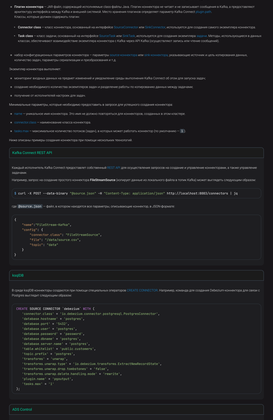
Кластер Kafka Connect состоит из компонентов **исполнителей** (worker).

Исполнитель (worker) - процесс, который запускает коннекторы и их задачи, а также управляет REST-API-запросами. Содержит основной параметр конфигурации с группой, данных, создает необходимые данные и балансировки нагрузки между исполнителями при добавлении или удалении воркеров. Исполнитель (worker) - процесс, который запускает коннекторы и их задачи, а также управляет REST-API-запросами. Для обеспечения масштабируемости и надежности Kafka Connect автоматически распределяет информацию между исполнителями. Если исполнитель останавливается или аварийно завершает работу, его задачи перезапускаются на других исполнителях. Если в группе исполнителей добавляются новые исполнители, они могут обнаружить существующие конфигурации коннекторов и задач и распределить нагрузку.

Конфигурация коннектора и задач хранится в специальных топиках Kafka, управляемых платформой **коннектора** (connector). Там же хранятся текущие статусы коннекторов и задач, включая последние смещения, отправленные в Kafka (или полученные из Kafka).

В процессе потоковой передачи Kafka Connect данные между данных в Kafka поддерживается периодическая фиксация данных. Платформа Kafka Connect может восстановиться после сбоя коннектора и продолжить обработку данных, даже если коннектор столкнулся с ошибкой.

Платформа Kafka Connect позволяет настраивать и управлять отдельными экземплярами платформы Kafka Connect. Основные компоненты Kafka Connect и как они взаимодействуют друг с другом:



Kafka Connect workers

Connector

Экземпляр коннектора (connector instance) - это логическая задача, которая планирование копирования отдельной и передает потоки в выходные потоки, при этом также отслеживает любые изменения в количестве задач, необходимых для копирования данных, и передает конфигурацию задач в Kafka Connect для последующего запуска исполнителя.

- **Source connector** - коннектор, который получает данные с API, затем генерирует данные, извлекает данные и схемы данных и передает их в Kafka Connect. Коннектор source могут отправлять данные Kafka для записи в источники, такие данные в топики Kafka при обновлении таблиц.
- **Sink connector** - коннектор, который получает данные из Kafka, затем отправляет их в другие системы, такие как индексы, такие как Elasticsearch, или в пакетные системы, такие как Hadoop для последующего анализа. Коннектор sink должен получать данные из Kafka и отправлять их в другие системы, такие как индексы, такие как Elasticsearch, или в пакетные системы, такие как Hadoop для последующего анализа.
- **Sink connector** - коннектор, который получает данные из Kafka, затем отправляет их в другие системы, такие как индексы, такие как Elasticsearch, или в пакетные системы, такие как Hadoop для последующего анализа.
- **connector class** - имя класса коннектора, например `io.confluent.connect.jdbc.JdbcSourceConnector`
- **connector type** - принадлежность класса коннектора, source или sink
- **serializer** - применяемая конфигурация данных, которая может работать с данными, например `io.confluent.connect.avro.AvroConverter`, `io.confluent.connect.protobuf.ProtobufConverter` или `org.apache.kafka.connect.storage.StringConverter`
- **transforms** - список операторов, которые применяются к каждому сообщению, которое создается или отправляется коннектором. Информация о том, как настроить данные, см. в разделе Kafka Connect Transformations.
- **key converter** - класс, используемый для сериализации ключей сообщений, которые коннектор записывает в Kafka или считывает из Kafka. Информация о том, как настроить данные, см. в разделе Kafka Connect Transformations.
- **value converter** - класс, используемый для сериализации значений сообщений, которые коннектор записывает в Kafka или считывает из Kafka. Информация о том, как настроить данные, см. в разделе Kafka Connect Transformations.
- **header converter** - класс, используемый для сериализации заголовков сообщений, которые коннектор записывает в Kafka или считывает из Kafka. Информация о том, как настроить данные, см. в разделе Kafka Connect Transformations.
- **task max** - максимальное количество задач, которое может создать этот коннектор. Задачи могут использовать меньше этого количества, если конфигурация коннектора не может быть разбита на столько задач. Если коннектор не может достичь этого уровня параллелизма, он может создать меньше задач или может быть полезно, если коннектор обрабатывает все данные в одной задаче. Значение по умолчанию - 1.

Также коннектор source содержит следующие свойства:

- **topic creation groups** - группы, используемые для определения топиков, которые должны быть созданы коннектором source. Дополнительную информацию см. в разделе Kafka Connect Source Connector Configuration Properties.
- **exactly.once.support** - чтобы разрешить коннектору source, должен ли коннектор source поддерживать семантику exactly-once. Допустимые значения: `requested` и `required`. Дополнительную информацию см. в разделе Kafka Connect Source Connector Configuration Properties.
- **transaction.boundary** - как часто коннектор source должен определять границы транзакций для данных, которые он создает. Допустимые значения: `poll`, `interval` и `connector`. Дополнительную информацию см. в разделе Kafka Connect Source Connector Configuration Properties.
- **offsets.storage.topic** - использовать ли коннектор source собственный топик для хранения смещений. Дополнительную информацию см. в разделе Kafka Connect Source Connector Configuration Properties.

Коннектор source также поддерживает параметры конфигурации, которые используются для настройки коннектора source. Например, коннектор JDBC source поддерживает параметры конфигурации, которые используются для настройки того, как подключаться к базе данных. Дополнительную информацию см. в разделе Kafka Connect JDBC Source Connector Configuration Properties.

Коннектор source также поддерживает параметры конфигурации, которые используются для настройки того, как подключаться к базе данных. Дополнительную информацию см. в разделе Kafka Connect JDBC Source Connector Configuration Properties.

Task

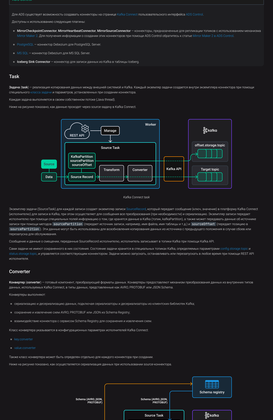
Задача (task) - реализация исполнения данных, которая запускает состояний в Kafka. Каждый экземпляр задачи создается из экземпляра коннектора при помощи конфигурации коннектора и выполняет часть работы коннектора. Задачи не хранят свое собственное состояние. Вместо этого состояние задачи хранится в специальных топиках Kafka (config, offset и status) и управляется связанным коннектором. Задачи могут быть запущены или остановлены в любое время, чтобы обеспечить масштабируемый конвейер потоковой передачи данных.

Также коннектор source содержит следующие свойства:

- **task.id** - идентификатор задачи, начиная с 0.
- **task.max** - максимальное количество задач, которое может быть создано этим коннектором source. Должно быть положительным числом.

Коннектор source также поддерживает параметры конфигурации, которые используются для настройки того, как подключаться к базе данных. Дополнительную информацию см. в разделе Kafka Connect JDBC Source Connector Configuration Properties.

Коннектор source также поддерживает параметры конфигурации, которые используются для настройки того, как подключаться к базе данных. Дополнительную информацию см. в разделе Kafka Connect JDBC Source Connector Configuration Properties.



Kafka Connect task

Экземпляр задачи (SourceTask) для каждой записи создает экземпляр записи source_record, который передает смещение (offset), значений и информацию Kafka Connect дополнительно для записи в Kafka. Для каждого экземпляра source_record создается экземпляр ConnectRecord, который содержит информацию о топике, разделе, ключе и значении. Экземпляр ConnectRecord передается в конвейер преобразования, где он преобразуется в соответствии с настроенными преобразованиями (transformations). Затем преобразованная запись source_record передается в конвертер (converter), который преобразует ее в двоичный формат, подходящий для записи в Kafka. Этот процесс зависит от настройки топиков.

Обзор Debezium

Ольга Семме

Содержание

Назад к содержанию

Возможности Debezium

Debezium и Kafka Connect

Особенности коннекторов Debezium для разных СУБД

- PostgreSQL
- MS SQL

Запуск коннектора Debezium

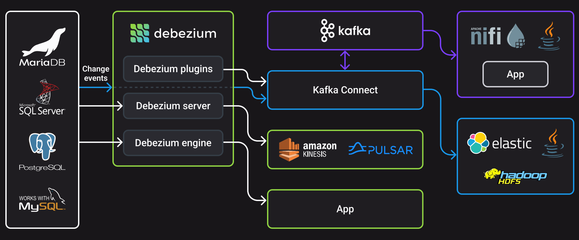
Возможности Debezium

Debezium — это распределенная платформа с открытым исходным кодом, работа которой основана на паттерне захвата изменения данных (Change Data Capture, CDC).

Коннектор Debezium собирает изменения в базах данных и передает их во внешние приложения.

Debezium используется внешними приложениями для учета и обработки изменения данных в СУБД: события вставки, обновления и удаления.

Ниже на рисунке показаны возможности платформы Debezium.



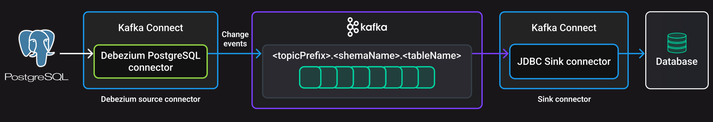
Возможности Debezium

Коннекторы Debezium могут быть запущены при помощи следующих способов:

- Через Kafka Connect с использованием готовых плагинов Debezium для создания коннекторов. При этом все изменения записываются в топики Kafka. Такой способ рекомендуется, если в системе используется потоковая передача событий в/из Kafka, чтобы исключить сторонних посредников.
- С использованием Debezium-сервера — готового приложения, которое передает события изменений из исходной базы данных в различные инфраструктуры обмена сообщениями.
- С использованием Debezium-движка, позволяющего встраивать коннекторы Debezium непосредственно в пространство приложения с использованием специального модуля *debezium-api*. При этом отсутствует такой уровень отказоустойчивости и надежности, как при использовании сервиса Kafka Connect, но в то же время отсутствуют такие посредники, как брокеры Kafka и сервис Kafka Connect.

Debezium и Kafka Connect

Ниже на рисунке показано, как выглядит архитектура конвейера сбора событий изменения при участии коннектора Debezium, например, при передаче событий из сервера PostgreSQL в любую базу данных, которая поддерживает JBDC.



Архитектура сбора измененных данных на основе Debezium

Коннектор Debezium, созданный при помощи Kafka Connect, записывает события изменений из PostgreSQL в топик Kafka, имя которого по умолчанию состоит из префикса, заданного пользователем, имени схемы базы данных и имени таблицы, из которой собираются изменения. После того, как записи о событиях изменений сохраняются в Kafka, различные sink-коннекторы, созданные Kafka Connect, могут передавать записи в другие базы и хранилища данных.

Debezium предоставляет набор коннекторов для передачи данных в топики Kafka из различных СУБД.

Особенности коннекторов Debezium для разных СУБД

Ниже описаны особенности работы коннекторов, доступных по умолчанию в ADS Control.

PostgreSQL

Захват изменения данных на сервере PostgreSQL обеспечивается при помощи механизма логической репликации на основе журнала упреждающей записи (Write Ahead Log, WAL). Этот лог хранится на диске, и в нем сохраняются все события изменения данных для запросов `INSERT`, `UPDATE` и `DELETE`. Эти изменения обрабатываются при помощи плагина вывода:

- decoderbufs — плагин, основанный на Protobuf;
- pgoutput — стандартный плагин вывода логического декодирования в PostgreSQL 10+, используется самим PostgreSQL для логической репликации.

Работа коннектора Debezium для PostgreSQL имеет ограничения, в том числе, связанные с используемой функцией логического декодирования PostgreSQL:

- Не поддерживаются изменения DDL — события, связанные с запросами `CREATE`, `ALTER`, `DROP`, `TRUNCATE`.
- Если PostgreSQL организован в виде кластера, коннектор может работать только с основным сервером, и, если он выходит из строя, коннектор останавливается. После восстановления основного сервера можно перезапустить коннектор. Если другой сервер PostgreSQL был повышен до основного, необходимо вручную настроить конфигурацию коннектора перед перезапуском.
- Модуль вывода логического декодирования `pgoutput` не фиксирует значения для сгенерированных столбцов.
- В настоящее время Debezium поддерживает базы данных только с кодировкой символов UTF-8. При однобайтовой кодировке символов невозможно правильно обрабатывать строки, содержащие символы расширенного кода ASCII.

Когда коннектор PostgreSQL впервые подключается к базе данных PostgreSQL, он создает согласованный моментальный снимок (снепшот) каждой из схем базы данных. Далее коннектор выполняет потоковую передачу изменений с того места, где был создан моментальный снимок.

Параметры потоковой репликации сервера PostgreSQL, важные для работы коннектора:

- `wal_level` — для включения поддержки логического декодирования необходимо установить в значение `logical`;
- параметры, влияющие на количество коннекторов, которые могут одновременно обращаться к серверу:
  - `max_replication_slots` — максимальное количество слотов репликации, которое может поддерживать сервер.
  - `max_wal_senders` — максимальное количество одновременных подключений от резервных серверов или клиентов потокового базового резервного копирования.
- `wal_keep_size` — максимальный размер WAL, который будет сохранять слот репликации.

Для настройки PostgreSQL Server для запуска коннектора Debezium при использовании плагина `pgoutput` требуется пользователь базы данных, который имеет следующие привилегии:

- `REPLICATION`
- `LOGIN`

Дополнительно при использовании плагина `pgoutput` пользователю требуются следующие привилегии:

- `CREATE` в базе данных для добавления публикаций.
- `SELECT` в таблицах для копирования исходных данных таблицы.

MS SQL

Чтобы разрешить коннектору Debezium для SQL Server фиксировать записи событий изменений для операций базы данных, необходимо сначала включить функцию отслеживания измененных данных, доступную начиная с SQL Server 2016. CDC необходимо включить в базе данных и в каждой таблице, которая должна быть захвачена.

Коннектор отслеживает операции `INSERT`, `UPDATE` и `DELETE` на уровне строк и записывает записи событий для каждой таблицы в отдельный топик Kafka.

При первом подключении коннектора к базе данных SQL Server он делает согласованный снимок схем (снепшот) для всех таблиц, указанных в его конфигурации, и передает его в Kafka. Далее коннектор непрерывно отслеживает происходящие изменения на уровне строк.

Запуск коннектора Debezium

Коннекторы Debezium, как и другие коннекторы, могут быть созданы при помощи нескольких способов через Kafka Connect.

На странице Kafka Connect пользовательского интерфейса ADS Control доступны к выбору плагины для создания некоторых коннекторов Debezium для передачи данных в топики Kafka кластера ADS, а также предоставляется возможность управлять работой коннекторов:

- PostgreSQL
- MS SQL

Остальные коннекторы могут быть установлены самостоятельно. Для этого необходимо создать JAR-файл, используя код коннектора, загруженный из репозитория Debezium, и сохранить его в папку, путь к которой указывается в качестве параметра ***plugin.path*** в группе ***connect-distributed.properties*** на странице конфигурирования сервиса ***Kafka Connect***.

Обзор Kafka REST Proxy

Ольга Семин

Содержание

Назад к содержанию

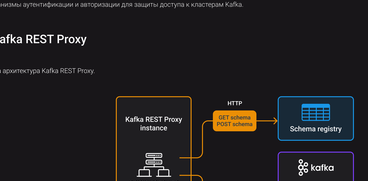
- Возможности Kafka REST Proxy
- Архитектура Kafka REST Proxy
  - HTTP-запросы к Kafka REST Proxy
  - Сериализация и десериализация
  - Безопасность
  - Ограничения Kafka REST Proxy
- Kafka REST Proxy в ADS
  - Подключение
  - Kafka
  - Schema Registry
  - Безопасность
  - Настройка

Возможности Kafka REST Proxy

Kafka REST Proxy — специальный прокси-сервер, созданный Confluent. Kafka REST Proxy предоставляет RESTful-интерфейс для взаимодействия с кластерами Kafka, позволяющий пользователям записывать и читать сообщения, используя HTTP/HTTPS-запросы.

Взаимодействие внешних систем с брокерами Kafka по умолчанию выполняется при помощи определенного набора API и инструментов CLI, обеспечиваемых клиентской библиотекой Java. Также поддерживаются клиентские библиотеки для некоторых языков программирования.

Kafka REST Proxy помогает обойти ограничения, связанные с узким выбором способов взаимодействия с Kafka. Любое приложение, в том числе основанное не на JVM, можно настроить для работы с кластером Kafka независимо от языка программирования, если оно имеет возможность отправлять HTTP/HTTPS-запросы.



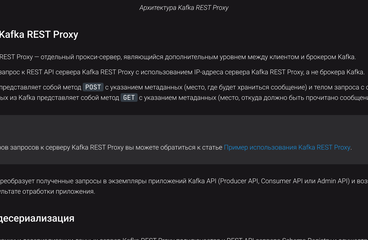
Возможности Kafka REST Proxy

Основные функциональные возможности Kafka REST Proxy перечислены ниже:

- Интерфейс HTTP RESTful позволяет клиентам взаимодействовать с Kafka при помощи стандартных методов HTTP, обеспечивая совместимость с различными языками программирования.
- Обеспечивается взаимодействие с Kafka API через прокси-сервер:
  - создание и отправка сообщений в топики Kafka в различных форматах;
  - подписка на топики и чтение сообщений в режиме реального времени;
  - получение данных о кластере и топиках.
- Поддерживается интеграция со Schema Registry для создания и развития схем данных.
- Поддерживаются механизмы аутентификации и авторизации для защиты доступа к кластерам Kafka.

Архитектура Kafka REST Proxy

Ниже на рисунке показана архитектура Kafka REST Proxy.



Архитектура Kafka REST Proxy

HTTP-запросы к Kafka REST Proxy

Каждый экземпляр Kafka REST Proxy — отдельный прокси-сервер, являющийся дополнительным уровнем между клиентом и брокером Kafka.

Клиент направляет HTTP-запрос к REST API сервера Kafka REST Proxy с использованием IP-адреса сервера Kafka REST Proxy, а не брокера Kafka.

Запрос на запись данных представляет собой метод `POST` с указанием метаданных (место, где будет храниться сообщение) и телом запроса с содержимым сообщения. Запрос на получение данных из Kafka представляет собой метод, `GET` с указанием метаданных (место, откуда должно быть прочитано сообщение).

> **ПРИМЕЧАНИЕ**
>
> Для получения примеров запросов к серверу Kafka REST Proxy вы можете обратиться к статье Пример использования Kafka REST Proxy.

Сервер Kafka REST Proxy преобразует полученные запросы в экземпляры приложений Kafka API (Producer API, Consumer API или Admin API) и возвращает пользователю ответ, основанный на результате отработки приложения.

Сериализация и десериализация

Для выполнения сериализации и десериализации данных сервер Kafka REST Proxy подключается к REST API сервера Schema Registry и осуществляет запись новой или получение уже имеющейся схемы. Для этого в пользовательских запросах к серверу Kafka REST Proxy используются специальные заголовки.

- `Content-Type` — определяет тип контента, который должен быть записан в Kafka.
- `Accept` — определяет тип содержимого, который может понять клиент.

В этих заголовках указывается тип контента в формате, использующемся в Schema Registry API.

Например, ниже приведен пример запроса, который записывает данные в топик Kafka, а также регистрирует схему в хранилище Schema Registry:

```
$ curl -X POST -H "Content-Type: application/vnd.kafka.avro.v2+json" \
      -H "Accept: application/vnd.kafka.v2+json" \
      --data '{"value_schema": "{\"type\": \"record\", \"name\": \"User\", \"fields\": [{\"name\": \"name\", \"type\": \"string\"}]}", "records": [{"value": {"name": "testUser"}}]}' \
      "http://localhost:8082/topics/avrotest"
```

Безопасность

На стороне сервера Kafka REST Proxy настраивается аутентификация для двух соединений:

- **Соединение между клиентом и сервером Kafka REST Proxy.**
  Для этого соединения настраивается базовая HTTP-аутентификация с использованием пары `<имя пользователя>:<пароль>`. Данные для аутентификации пользователя передаются Kafka REST Proxy в HTTP-запросе при помощи заголовка `Authorization`.

  Для включения взаимной SSL/TLS-аутентификации настраивается:

  - поддержка HTTPS для сервера Kafka REST Proxy;
  - хранилище доверенных сертификатов Kafka REST Proxy для проверки входящих клиентских сертификатов.

- **Соединение между Kafka REST Proxy и брокерами Kafka.**
  Если Kafka REST Proxy используется для связи с брокером Kafka, в котором уже настроена аутентификация по методу SASL, для каждого сервера Kafka REST Proxy необходимо установить соответствующие параметры для аутентификации пользователя Kafka REST Proxy в Kafka.

  Если для брокера Kafka настроена SSL/TLS-аутентификация, для аутентификации пользователя Kafka REST Proxy необходимо настроить SSL-параметры.

Ограничения Kafka REST Proxy

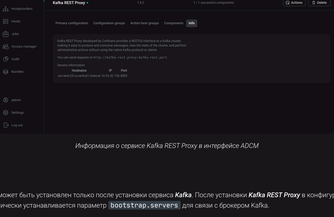
При использовании Kafka REST Proxy в системах обработки данных необходимо учитывать следующие ограничения:

- Не поддерживаются запросы на запись сообщений в несколько топиков одновременно — каждый HTTP-запрос от производителя может обращаться лишь к одному топику или партиции.
- При помощи HTTP-запросов недоступно большинство переопределений конфигураций производителей и потребителей.
- Недоступно использование разных сериализаторов для ключа и значения в сообщении.
- Обработка запросов к Kafka REST Proxy производится при помощи синхронного протокола запроса-ответа. При этом для Kafka возможно подключение до десятков параллельных экземпляров Kafka REST Proxy, что достаточно для разных вариантов использования.

Kafka REST Proxy в ADS

Подключение

После добавления и установки сервиса ***Kafka REST Proxy*** в составе кластера ADS в запросе к серверу Kafka REST Proxy должен быть указан хост, где размещен компонент ***Kafka REST Proxy***, и порт, указанный в качестве параметра ***rest_listener_port*** (по умолчанию `8082`) в группе ***Main*** на странице конфигурирования сервиса ***Kafka REST Proxy***.



Информация о сервисе Kafka REST Proxy в интерфейсе ADCM

Kafka

В ADS сервис ***Kafka REST Proxy*** может быть установлен только после установки сервиса ***Kafka***. После установки ***Kafka REST Proxy*** в конфигурационном файле */etc/kafka-rest/kafka-rest.properties* автоматически устанавливается параметр `bootstrap.servers` для связи с брокером Kafka.

Schema Registry

Сервис ***Schema Registry***, установленный в ADS одновременно с сервисом ***Kafka REST Proxy***, позволяет выполнять сериализацию и десериализацию данных при работе с топиками Kafka по интерфейсу RESTful. После установки обоих сервисов параметр `schema.registry.url` автоматически устанавливается в конфигурационном файле Kafka REST Proxy */etc/kafka-rest/kafka-rest.properties*.

Безопасность

Большинство опций безопасности для Kafka REST Proxy могут быть настроены при помощи интерфейса ADCM.

- В кластере ADS базовую HTTP-аутентификацию для Kafka REST Proxy можно настроить при помощи включения Basic-аутентификации.
- В кластере ADS аутентификация по методу SASL включается при помощи действия Manage Kerberos либо при запуске мастера SASL PLAINTEXT.
- В кластере ADS специальный модуль kafka-rest-security предоставляет возможность проходить аутентификацию в Kafka с разными принципалами при подключении к Kafka через Kafka REST Proxy после включения аутентификации по методу SASL.
- В кластере ADS SSL-аутентификация включается при помощи действия Manage SSL.

Настройка

Настройка параметров Kafka REST Proxy в интерфейсе ADCM выполняется на странице конфигурирования сервиса ***Kafka REST Proxy***.

Для настройки параметров конфигурационных файлов */etc/kafka-rest/kafka-rest.properties* переведите в активное состояние переключатель ***Show advanced***, раскройте узел ***kafka-rest.properties*** или ***connect.properties*** и введите новые значения для параметров. Для изменения параметров ksqlDB, отсутствующих в интерфейсе ADCM, используйте поле ***Add key:value***. Выберите ***Add property*** и введите наименование параметра и его значение.

После изменения параметров при помощи интерфейса ADCM перезагрузите сервис ***Kafka REST Proxy***. Для этого примените действие Restart, нажав на иконку в столбце ***Actions***.

Обзор ksqlDB

Ольга Семме

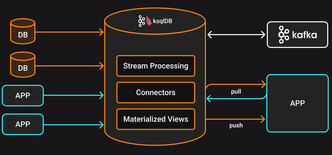
Содержание

Назад к содержанию

- Возможности ksqlDB
- Архитектура ksqlDB
- ksqlDB в ADS
  - Подключение
  - Kafka
  - Schema Registry
  - Kafka Connect
  - Настройка

Возможности ksqlDB

ksqlDB – это база данных Kafka, использующаяся для потоковой обработки. ksqlDB обрабатывает данные, хранящиеся в Kafka, и объединяет Kafka Streams для потоковой обработки и Kafka Connect для сбора и обработки событий из различных источников данных. Приложения потоковой обработки Kafka Streams внутри ksqlDB являются набором SQL-запросов.



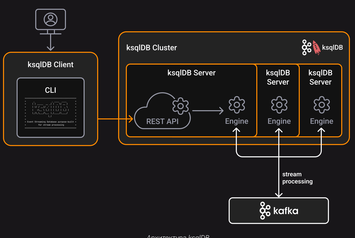
Возможности ksqlDB

Основные функциональные возможности ksqlDB перечислены ниже:

- Моделирование данных, хранящихся в Kafka, в виде потоков или таблиц посредством SQL.
- Выполнение push-запросов на передачу данных. При помощи push-запросов можно осуществлять подписку на изменения, происходящие в базе данных, и отслеживать изменения в режиме реального времени. Push-запрос может быть отправлен через интерфейс командной строки (CLI) или как HTTP-запрос в REST API ksqlDB.
- Выполнение pull-запросов на получение данных. Pull-запрос может извлекать текущее значение из материализованного представления, таблицы или потока.
- Создание материализованных представлений из потоков и таблиц. Материализованное представление в фоновом режиме собирает данные из таблицы Kafka, преобразует их в необходимый формат и помещает в заранее созданную целевую таблицу. К одной таблице Kafka можно привязать несколько материализованных представлений, чтобы сохранять данные с разным уровнем детализации в несколько таблиц.
- Создание коннекторов для интеграции с внешними хранилищами данных.

Архитектура ksqlDB

Ниже на рисунке показана архитектура ksqlDB.



Архитектура ksqlDB

Каждый сервер ksqlDB осуществляет выполнение приложений Kafka Streams и является, по сути, отдельным экземпляром приложения Kafka Streams. В составе кластера несколько серверов ksqlDB делят между собой нагрузку, созданную топологией Kafka Streams.

Сервер ksqlDB состоит из двух компонентов:

- **Движок ksqlDB** – выполняет операторы и запросы SQL. Движок производит синтаксический анализ операторов SQL, преобразование в топологию Kafka Streams, запуск приложений Kafka Streams.
- **REST API ksqlDB** – обеспечивает доступ клиента к движку ksqlDB для передачи запросов.

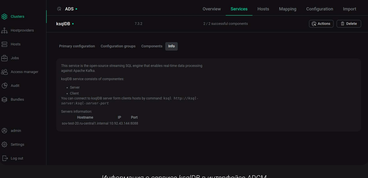
Для подключения к интерфейсу REST ksqlDB клиент может использовать следующие виды интерфейса ksqlDB:

- интерфейс командной строки ksqlDB CLI – консоль, в которой вводятся SQL-запросы с использованием операторов ksqlDB;
- пользовательский интерфейс ksqlDB – позволяет разрабатывать приложения ksqlDB в Центре управления Confluent и Confluent Cloud;
- непосредственно сам интерфейс REST – позволяет осуществлять запросы к серверу при помощи ksqlDB REST API с использованием инструмента командной строки cURL.

ksqlDB в ADS

Подключение

После добавления и установки сервиса ***ksqlDB*** в составе кластера ADS вы можете подключиться к серверу ksqlDB с хостов, где размещен компонент ***ksqlDB Client*** по команде: `ksql http://ksql-server:ksql-server-port`



Информация о сервисе ksqlDB в интерфейсе ADCM

Kafka

В ADS сервис ***ksqlDB*** может быть установлен только после установки сервиса ***Kafka***. После установки ***ksqlDB*** в конфигурационном файле */etc/ksqldb/ksql-server.properties* автоматически устанавливается параметр `bootstrap.servers` для связи с брокером Kafka, а также другие параметры для взаимодействия Kafka и ksqlDB (например, для внутренних топиков ksqlDB, создаваемых в Kafka).

Schema Registry

Сервис ***ksqlDB***, установленный в ADS одновременно с сервисом Schema Registry, позволяет выполнять регистрацию, чтение схем, а также сериализацию данных с использованием указанных (при помощи идентификатора) схем. Это облегчает работу с сериализацией данных, так как нет необходимости определять в ksqlDB столбцы и типы данных вручную. Формат может быть создан как для ключей, так и для значений. Например, при создании потока на основе топика Kafka использование схем описывается свойствами `KEY_SCHEMA_ID` или `VALUE_SCHEMA_ID` потока, как показано ниже:

```
CREATE STREAM pageviews
  WITH (
    KAFKA_TOPIC='avro-topic',
    VALUE_FORMAT='AVRO',
    VALUE_SCHEMA_ID=1
  );
```

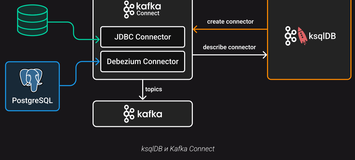
В ksqlDB могут быть указаны столбцы, содержимое которых не соответствует формату схемы. При этом может быть сохранен вывод столбцов, соответствующих формату схемы.

После установки сервиса ***ksqlDB*** параметры `key.converter.schema.registry.url` и `value.converter.schema.registry.url`, отвечающие за взаимодействие ksqlDB и Schema Registry, устанавливаются в конфигурационном файле */etc/ksqldb/connect.properties*.

Kafka Connect

Сервис ***ksqlDB***, установленный в ADS одновременно с сервисом ***Kafka Connect***, предоставляет возможность управлять коннекторами Kafka Connect, выполняя действия:

- создание коннекторов;
- описание коннекторов;
- удаление коннекторов;
- импорт топиков, созданных Kafka Connect, в собственную базу для использования в потоковой обработке.



ksqlDB и Kafka Connect

Примеры коннекторов, которые могут быть созданы при помощи ksqlDB:

- Коннектор Debezium PostgreSQL – source-коннектор, который получает снепшот из базы данных PostgreSQL, затем отслеживает все последующие изменения этих данных на уровне строк для каждой таблицы и записывает в отдельный топик Kafka;
- JDBC-коннекторы – коннекторы, работающие с любой базой данных с драйвером JDBC с импортом данных в топики Kafka (коннектор JDBC Source) или экспортом данных из топиков Kafka (коннектор JDBC Sink).

Для взаимодействия сервисов ***ksqlDB*** и ***Kafka Connect*** необходимо настроить параметр `ksql.connect.url` в конфигурационном файле */etc/ksqldb/connect.properties*.

Настройка

Настройка параметров ksqlDB в интерфейсе ADCM выполняется на странице конфигурирования сервиса ***ksqlDB***.

Для настройки параметров конфигурационных файлов */etc/ksqldb/ksql-server.properties* и */etc/ksqldb/connect.properties* переведите в активное состояние переключатель ***Show advanced***, раскройте узел ***server.properties*** или ***connect.properties*** и введите новые значения для параметров. Для изменения параметров ksqlDB, отсутствующих в интерфейсе ADCM, используйте поля ***Add key:value***. Выберите ***Add property*** и введите наименование параметра и его значение.

После изменения параметров при помощи интерфейса ADCM перезагрузите сервис ***ksqlDB***. Для этого примените действие Restart, нажав на иконку в столбце ***Actions***.

Обзор MiNiFi

Ольга Семме

Содержание

Назад к содержанию

- Возможности MiNiFi
- Архитектура
  - C2 Server
  - MiNiFi Agent
- Реализация MiNiFi
- MiNiFi в ADS

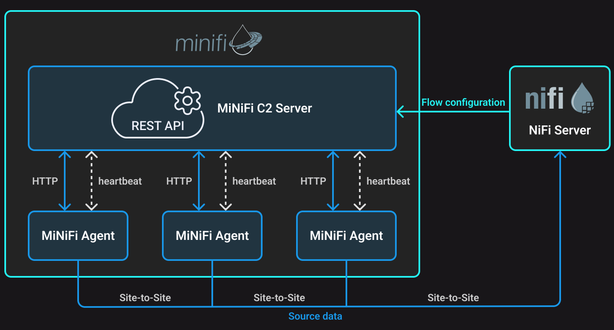
Возможности MiNiFi

MiNiFi — подпроект, дополняющий функциональные возможности NiFi. MiNiFi предоставляет возможность сбора данных в наиболее удаленных периферийных устройствах, где невозможно установить полноценный NiFi Server, например: вышки сотовой связи, устройства доступа к сети, маршрутизаторы и т.д. Сбор данных непосредственно у источника снижает вероятность потери данных при передаче.

Для обеспечения легковесности в MiNiFi отсутствуют многие функции, присутствующие в NiFi. Так, например, из-за отсутствия пользовательского интерфейса в MiNiFi построение потока данных может быть осуществлено только в полноценном NiFi для создания шаблонов, которые будут использованы в MiNiFi.

Архитектура

Ниже на рисунке показана архитектура MiNiFi.



Архитектура MiNiFi

Работа MiNiFi основана на построении настраиваемого потока данных NiFi. Множество агентов (MiNiFi Agent) собирают данные от датчиков или других устройств (например, через протокол MQTT) и передают их в поток при помощи специально настроенных процессоров (например, ConsumeMQTT). После создания потока данных и добавления необходимых атрибутов для FlowFile (параметры датчика, локация и т.д.) при помощи удаленной группы процессов (Remote Process Group) данные отправляются в основной экземпляр NiFi.

C2 Server

MiNiFi C2 Server — центральный сервер, управляющий конфигурацией и работой множества агентов через протокол C2. MiNiFi C2 Server устанавливается на сервер, который должен быть доступен с агентов MiNiFi. Может быть использовано несколько серверов C2 для сложных сетевых структур.

Ниже описаны основные функции MiNiFi C2 Server:

- Сервер отслеживает heartbeat-сообщения от агентов и в ответ при необходимости отправляет команды на обновление или запись конфигурации потока.
- У каждого нового агента сервер запрашивает манифест (все предварительно установленные конфигурации, процессоры и службы контроллера). Манифест агента используется в протоколе C2 для проектирования потока для класса агента — группы агентов с одинаковыми манифестами.
- Сервер управляет конфигурациями потоков MiNiFi и публикует их через собственный REST API. Агенты могут соединяться с этим API для обновления собственной конфигурации через периоды времени, определяемые параметром `nifi.minifi.notifier.ingestors.pull.http.period.ms`.
  Конфигурация потока представляет собой YML-файл (*config.yml*) и может быть создана вручную. Так как этот способ является трудоемким, наиболее часто используется поток, созданный при помощи пользовательского интерфейса NiFi и сохраненный как шаблон в формате XML-файла. Доступ MiNiFi C2 Server к шаблонам осуществляется через NiFi REST API. В YML-формат шаблон может быть преобразован при помощи утилиты `minifi-toolkit`. Имя шаблона, как правило, соответствует имени класса агента.

MiNiFi Agent

MiNiFi Agent — агент, запускающий потоки данных и представляющий, по сути, облегченную версию NiFi.

Ниже описаны основные функции MiNiFi Agent:

- В ответ на heartbeat-сообщения агент получает от сервера C2 команду на обновление конфигурации.
- Агенты отправляют подтверждение всех управляющих команд, полученных от сервера по протоколу C2.
- Агент загружает поток, используя REST API сервера. Для указания класса агента используется параметр `nifi.minifi.notifier.ingestors.pull.http.query`.
- Агент применяет новый поток и отправляет собранные данные по протоколу Site-to-Site на экземпляр удаленного NiFi Server, используя удаленную группу процессов (Remote Process Group).

Реализация MiNiFi

MiNiFi доступна в двух вариантах:

- MiNiFi C++ — представляет собой двоичный файл, потребляет мало системной памяти, но может запускать ограниченное подмножество процессоров NiFi.
- MiNiFi Java — содержит большинство доступных процессоров NiFi, но представляет собой более крупный двоичный дистрибутив и потребляет больше системных ресурсов.

MiNiFi в ADS

Сервис ***MiNiFi*** в кластере ADS может быть установлен только одновременно с сервисом ***NiFi*** или при условии, что ***NiFi*** уже установлен.

После добавления и установки сервиса ***MiNiFi*** в составе кластера ADS для каждого компонента сервиса автоматически устанавливаются параметры:

- Для компонента ***MiNiFi C2 Server*** в файле */etc/minifi-c2/conf/minifi-c2-context.xml* устанавливается адрес API экземпляра NiFi, установленного в этом же кластере ADS, при помощи которого будут загружены шаблоны потоков (`http://<hostname>:9090/nifi-api`).
- Для компонента ***MiNiFi Agent*** в файле */etc/minifi/conf/bootstrap.conf* автоматически устанавливаются параметры для обновления конфигураций потоков и связи с MiNiFi C2 Server, а также загружается текущая конфигурация потока (если она имеется) в файл */etc/minifi/conf/config.yml*.

Настройка параметров и изменение параметров, заданных по умолчанию, выполняется на странице конфигурирования сервиса ***MiNiFi*** в интерфейсе ADCM.

Для подключения к экземпляру NiFi по протоколу Site-to-Site для сервиса ***NiFi*** установлены параметры, отвечающие за удаленное подключение (с префиксом `nifi.remote`) в файле */etc/nifi/conf/nifi.properties*.

Описания параметров приведены в разделах ***MiNiFi*** и ***NiFi*** статьи Конфигурационные параметры ADS.

После изменения параметров при помощи интерфейса ADCM перезагрузите сервис. Для этого примените действие ***Restart***, нажав на иконку в столбце ***Actions***.

Обзор NiFi

Ольга Семин

Содержание

Назад к содержанию

- Функции
- Архитектура
  - Основные компоненты узла NiFi
  - Архитектура кластера NiFi
- Настройка NiFi

Функции

NiFi – платформа, созданная для автоматизации управления потоками данных между системами.

Основные функциональные возможности NiFi перечислены ниже:

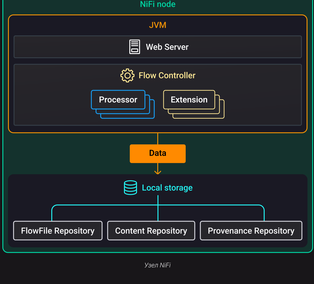
- Управление потоками данных при помощи использования специальных процессоров, каждый из которых имеет свое функциональное назначение. При этом для обеспечения доступности данных, гарантии доставки данных и устойчивости при обработке перезагрузок и неожиданных сбоев системы поддерживаются следующие функции:
  - Использование **журнала упреждающей записи (Write-Ahead Log)**, который регистрирует каждое изменение, происходящее с объектом потока данных (FlowFile) в качестве транзакционной единицы и фиксирует успешное изменение метаданных в репозитории FlowFile, в случае сбоя системы восстанавливает свое состояние.
  - Возможность гибкой настройки буферизации данных в очереди с применением backpressure при достижении очередью определенных параметров.
  - Возможность настройки схем приоритизации обработки данных из очереди.
- Функции, обеспечивающие простоту управления потоками:
  - Пользовательский интерфейс NiFi предоставляет возможность визуально создавать потоки данных и управлять ими в режиме реального времени. Для создания новых компонентов в системе или внесения изменения в существующие нет необходимости останавливать все компоненты.
  - Шаблоны потоков позволяют создавать и обмениваться проектами потоков.
  - Хранение в репозитории Data Provenance информации о происхождении данных и всех операций, которым они были подвергнуты при прохождении через систему. Это позволяет проводить анализ системы на соответствие заданным требованиям, а также оптимизацию и устранение неполадок.
  - Возможность просмотра контента, сохраненного в репозитории контента в любой момент цикла обработки потока.
- Функции, обеспечивающие безопасность системы:
  - Возможность использования протоколов с шифрованием (двусторонний SSL) при передачи данных между двумя системами.
  - NiFi поддерживает аутентификацию SSL и обеспечивает подключаемую авторизацию в пользовательском интерфейсе или при помощи Apache Ranger. Также может быть настроена **мультитенантная авторизация**, позволяющая проводить настройку для групп пользователей доступа к разным частям потока данных с различными уровнями авторизации.
- Возможности расширений:
  - NiFi предоставляет возможность подключения расширений через следующие точки: процессоры, службы контроллера, задачи отчетности, приоритизаторы и пользовательские интерфейсы клиентов.
  - Ограниченный набор зависимостей для создания расширений и применение собственной модели загрузчика классов NiFi.
  - Возможность соединения между экземплярами NiFi при помощи протокола связи **Site-to-Site (S2S)**, что обеспечивает безопасную и эффективную передачу данных, а также использование библиотек расширений для передачи данных.
- Масштабирование:
  - NiFi осуществляет масштабирование за счет кластеризации нескольких узлов. Это позволяет настраивать балансировку потоков и аварийное переключение между узлами.
  - Регулирование пропускной способности может быть выполнено при помощи изменения количества задач на вкладке SCHEDULING при настройке процессора.

Архитектура

Основные компоненты узла NiFi

NiFi выполняется внутри JVM в операционной системе хоста.

Компоненты узла NiFi представлены на следующей схеме:



Узел NiFi

Узел NiFi содержит следующие компоненты:

- **Веб-сервер (Web Server)** – сервер, принимающий HTTP-запросы от клиентов для управления потоками NiFi и выдающий результат в виде демонстрации изменений потока визуально (при помощи пользовательского интерфейса) или HTTP-ответов.
- **FlowFile** – объект NiFi, представляющий собой информационный пакет, перемещающийся через систему. Для каждого пакета NiFi отслеживает атрибуты в виде пар ключ/значение и связанное с пакетом содержимое (content).
- **Процессор (Processor)** – компонент NiFi, выполняющий заявленные в своей функции действия, связанные с обработкой данных FlowFiles, например, извлечение или публикацию данных. Процессоры могут использовать один или несколько FlowFiles. Процессоры имеют доступ к атрибутам данного FlowFile и его содержимого.
- **Расширения (Extension)** – библиотеки, увеличивающие или изменяющие функционал объекта и загружаемые во время запуска точек расширения. Расширения в NiFi запускаются и выполняются внутри JVM. Точки расширения включают в себя: процессоры, службы контроллера, задачи отчетности, приоритизаторы и пользовательские интерфейсы клиентов.
- **Контроллер потока (Flow Controller)** – компонент NiFi, выполняющий функции управления обменом FlowFiles между процессорами и управление запуском расширений. Контроллер потока может быть настроен для управления определенным набором потоков.
- **Репозитории NiFi** — репозитории, хранящие данные для работы потоков NiFi. Репозитории представляют собой каталоги в локальном хранилище узла NiFi:
  - **FlowFile Repository** – репозиторий, который содержит метаданные для всех текущих FlowFile в потоке, отслеживающие весь путь прохождения FlowFile в системе. Реализация репозитория является подключаемой. По умолчанию для хранения данных используется журнал упреждающей записи. Путь расположения репозитория на диске указывается при помощи параметра `nifi.flowfile.repository.directory`
  - **Content Repository** – репозиторий, в котором содержатся фактические байты содержимого данного FlowFile. Реализация репозитория является подключаемой. По умолчанию репозиторий представляет собой механизм, который хранит блоки данных в файловой системе. Путь расположения репозитория на диске указывается при помощи параметра `nifi.content.repository.directory`. Можно указать более одного места хранения файловой системы.
  - **Provenance Repository** – репозиторий, в котором содержатся все данные о происхождении событий, история FlowFiles. Данные о событиях в каждом месте индексируются и доступны для поиска. Реализация репозитория является подключаемой. Путь расположения репозитория на диске указывается при помощи параметра `nifi.provenance.repository.directory`. Можно указать один или несколько томов физического диска.

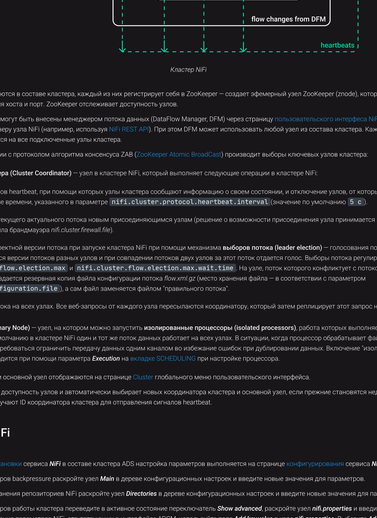
Архитектура кластера NiFi

Начиная с версии 1.0 в NiFi используется парадигма **кластеризации с нулевым лидером (Zero-Leader Clustering)**, в которой нет единого лидера системы, и соответственно, нет единой точки отказа.

Для координации работы узлов NiFi в распределенной системе (в кластере) используется ZooKeeper, который обеспечивает доступность данных потока в случае сбоев или технического обслуживания узлов.

> **ПРИМЕЧАНИЕ**
>
> - Концепции и основные компоненты ZooKeeper описаны в статье ZooKeeper.
> - Механизм выбора лидера при помощи ZooKeeper описан в статье Выбор лидера в ZooKeeper.

На рисунке ниже показано, как взаимодействуют узлы NiFi друг с другом и с ZooKeeper в кластере.



Кластер NiFi

Когда узлы NiFi запускаются в составе кластера, каждый из них регистрирует себя в ZooKeeper – создает эфемерный узел ZooKeeper (znode), который указывает идентификатор узла, имя хоста и порт. ZooKeeper отслеживает доступность узлов.

Изменения в потоке NiFi могут быть внесены менеджером потока данных (DataFlow Manager, DFM) через страницу пользовательского интерфейса NiFi или при помощи подключения к веб-серверу узла NiFi (например, используя NiFi REST API). При этом DFM может использовать любой узел из состава кластера. Каждое внесенное изменение реплицируется на все подключенные узлы кластера.

ZooKeeper в соответствии с протоколом алгоритма консенсуса ZAB (ZooKeeper Atomic BroadCast) производит выборы ключевых узлов кластера:

- **Координатор кластера (Cluster Coordinator)** – узел в кластере NiFi, который выполняет следующие операции в кластере NiFi:
  - Обработка сигналов heartbeat, при помощи которых узлы кластера сообщают информацию о своем состоянии, и отключение узлов, от которых не поступает сигнал heartbeat в течение времени, указанного в параметре `nifi.cluster.protocol.heartbeat.interval` (значение по умолчанию `5 с`).
  - Предоставление текущего актуального потока новым присоединяющимся узлам (решение о возможности присоединения узла принимается на основе настроенного файла брандмауэра *nifi.cluster.firewall.file*).
  - Определение корректной версии потока при запуске кластера NiFi при помощи механизма **выборов потока (leader election)** — голосования по потокам узлов. При этом сравниваются версии потоков разных узлов и при совпадении потоков двух узлов за этот поток отдается голос. Выборы потока регулируются параметрами `nifi.cluster.flow.election.max` и `nifi.cluster.flow.election.max.wait.time`. На узле, поток которого конфликтует с потоком "победившего на выборах" узла, создается резервная копия файла конфигурации потока *flow.xml.gz* (место хранения файла — в соответствии с параметром `nifi.flow.configuration.file`), а сам файл заменяется файлом "правильного потока".
  - Согласование потока на всех узлах. Все веб-запросы от кластера узла пересылаются координатору, который затем реплицирует этот запрос на все узлы кластера.
- **Основной узел (Primary Node)** – узел, на котором можно запустить **изолированные процессоры (isolated processors)**, работа которых выполняется только на одном узле кластера. По умолчанию в кластере NiFi один и тот же поток данных работает на всех узлах. В ситуации, когда процессор обрабатывает файлы из удаленного каталога, может потребоваться ограничить передачу данных одним каналом во избежание ошибок при дублировании данных. Включение "изолированности" процессора производится при помощи параметра **Execution** на вкладке SCHEDULING при настройке процессора.

Координатор кластера и основной узел отображаются на странице Cluster глобального меню пользовательского интерфейса.

ZooKeeper отслеживает доступность узлов и автоматически выбирает новых координатора кластера и основной узел, если прежние становятся недоступными. Также из ZooKeeper все узлы получают ID координатора кластера для отправления сигналов heartbeat.

Настройка NiFi

После добавления и установки сервиса **NiFi** в составе кластера ADS настройка параметров выполняется на странице конфигурирования сервиса **NiFi** в интерфейсе ADCM.

Для настройки параметров backpressure раскройте узел **Main** в дереве конфигурационных настроек и введите новые значения для параметров.

Для изменения мест хранения репозиторием NiFi раскройте узел **Directories** в дереве конфигурационных настроек и введите новые значения для параметров.

Для настройки параметров работы кластера переведите в активное состояние переключатель **Show advanced**, раскройте узел **nifi.properties** и введите новые значения для параметров. Для изменения параметров NiFi, отсутствующих в интерфейсе ADCM, используйте поле **Add key,value** в узле **nifi.properties**. Выберите **Add property** и введите наименование параметра и его значение.

Обзор Schema Registry

Илья Клебанов

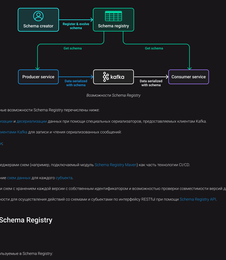
Содержание

Назад к содержанию

- Возможности Schema Registry
- Архитектура Schema Registry
  - Обзор
  - Кластер Schema Registry
  - Производители
  - Потребители
- Schema Registry в ADS
  - Подключение
  - Kafka
  - ksqlDB
  - Kafka Connect
  - Kafka REST Proxy
  - NiFi
  - Настройка Schema Registry

Возможности Schema Registry

Schema Registry — централизованный репозиторий, используемый для обеспечения согласованности формата данных между производителями и потребителями сообщений в Kafka.



Возможности Schema Registry

Основные функциональные возможности Schema Registry перечислены ниже:

- Обеспечение [сериализации и десериализации](#) данных при помощи специальных сериализаторов, предоставляемых клиентам Kafka.
- Взаимодействие с клиентами Kafka для записи и чтения сериализованных сообщений:
  - с производителями;
  - с потребителями;
  - с отдельным менеджером схем (например, поддерживаемый проектом Schema Registry Maven) как часть технологии CI/CD.
- Регистрация и хранение [схем данных](#) для каждого [субъекта](#).
- Поддержка эволюции схем с хранением каждой версии с собственным идентификатором и возможностью проверки совместимости версий для каждого субъекта.
- Возможность для клиентов осуществлять действия со схемами и субъектами по интерфейсу RESTful при помощи Schema Registry API.

Архитектура Schema Registry

Обзор

Основные понятия, используемые в Schema Registry.

- **Сериализация** — процесс записи данных для формата структуры, специальным поставщиком по типам/классам библиотеки (или кода) в формате двоичного кода для передачи объектов. Schema Registry поддерживает следующие форматы сериализации: JSON, AVRO, PROTOBUF.
  Преимущества сериализации:
  - возможность использовать различные форматы данных в сообщениях Kafka;
  - возможность обмена данными между приложениями, разработанными на разных языках программирования;
  - уменьшение объёма сохраняемых данных.
- **Десериализация** — восстановление данных из потока байтов.
- **Сериализатор и десериализатор** — это объекты (или классы), которые представляют собой набор функций и методов, обеспечивающих взаимодействие клиента со Schema Registry для обеспечения сериализации и десериализации данных. Существуют готовые встроенные сериализаторы и десериализаторы Kafka для ключа и формата схем. Kafka API поддерживает получение и использование сериализаторов и десериализаторов из конфигурации [Serializer]. Сериализатор определяется в конфигурации производителя так: `key.serializer` или `value.serializer`. Десериализатор определяется в конфигурации потребителя при помощи параметров `key.deserializer` или `value.deserializer`.
- **Субъект** — область, в которой могут развиваться схемы. Имя субъекта создается при передаче сериализатора схемы и используется:
  - при извлечении последней схемы или при проверке схемы для различных версий;
  - для проверки совместимости версий схем;
  - для получения схемы из хранилища.

  Существует три наиболее часто используемых способа присвоения имени субъекту:
  - `TopicNameStrategy` — настройка по умолчанию. Схемы, которые регистрируются производителем, будут зарегистрированы под именем топика с добавлением `-key` или `-value`, например, для топика `topic` ключ и значение будут иметь значения субъекта `topic-key` и `topic-value`.
  - `RecordNameStrategy` — позволяет использовать схемы для одного топика, поскольку каждая отдельная запись будет в своей схеме. При этом проверка использования схем для чего-то конкретного идет на уровне всего кластера, а не отдельного топика, что может быть полезно, если несколько приложений, с каждым своим уникальным набором схем.
  - `TopicRecordNameStrategy` — представляет собой комбинацию первых двух стратегий.

  Имя субъекта, отличное от заданного по умолчанию, устанавливается при помощи параметра производителя для регистрации новой схемы: `key.subject.name.strategy` или `value.subject.name.strategy`.
- **Схема** — структура формата данных, поддерживаю сериализацию. Схема описывает данные, записываемые в топик, какой тип информации он содержит. Это обеспечивает связывание производителя и потребителей. Существует несколько форматов данных, например, схема формата в данных соответствовать [представлена в формате AVRO].

  Пример простейшей схемы содержащей информацию для одного и того же образца использования данных: `{"id":"1000", "amount":500 }`

JSON

```
{
  "type": "object",
  "properties": {
    "id": {
      "type": "string"
    },
    "amount": {
      "type": "number"
    }
  }
}
```

AVRO

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

PROTOBUF

```
syntax = "proto3";

message MyRecord {
    string id = 1;
    float amount = 2;
}
```

- **Совместимость схемы** — соответствие новой версии схемы предыдущим версиям для данного субъекта. Могут быть определены несколько типов совместимости и изменяются от того, какие данные были изменены или удалены. Типы совместимости включают: по типу, удобочтение того, как может воспользоваться схема, чтобы все новое версии успешно могли быть обработаны производителем и потребителем. Тип совместимости определяется при помощи Schema Registry Kafka compatibility.conf, который называется в файле конфигурации.

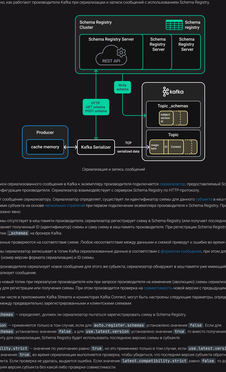
Кластер Schema Registry

Каждый сервер Schema Registry является, по сути, сервером REST API. Сервисы Schema Registry предоставляют API для управления схемами, а также производителю и десериализатора для чтения и записи данных в соответствии со схемами.

Все серверы Schema Registry в составе кластера ADS равнозначно осуществляют обслуживание процессов операционных и имеют доступ к хранилищу схем. В производственных средах рекомендуется указывать URL всех серверов Schema Registry.

Производители

Ниже на рисунке показано, как работают производители Kafka при сериализации и записи сообщений с использованием Schema Registry.



Сериализация и запись сообщений

Для сериализации и записи сериализованного сообщения в Kafka с экземпляру производителя подключается [сериализатор], предоставленный Schema Registry, тип которого определяет конфигурация производителя. Сериализатор обращается к Schema Registry по HTTP-протоколу.

Производитель передает сообщение сериализатору. Сериализатор определяет, существует ли идентификатор схемы для данного [субъекта] и если нашего производителя. Сериализатор получает имя субъекта из основе [стратегии субъекта], при первом подключении производителя к Schema Registry.

Перед сериализацией данные проверяются на соответствие схеме. Любое несоответствие между данными и схемой приведут к ошибке во время сериализации.

После получения ID схемы производитель записывает в топик Kafka сериализованное сообщение в [формате сообщения], при этом добавляет перед ними специальный магик-байт (magic byte) версии формата сериализации.

Если тот же экземпляр производителя сериализует новое сообщение для этого же субъекта, сериализатор обращается в кэш-память уже имеющихся ID схемы и схему и в Schema Registry не отправляет.

Для записи сообщения в новый топик при производителем или при запуске производителя на изменение (эволюцию) схемы сериализатор выполняет новый запрос к Schema Registry для регистрации схемы. Если схема не зарегистрирована, то она регистрируется и производителю [новый] версию и продолжает.

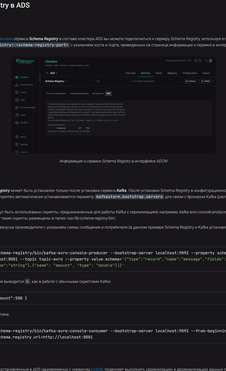
Для производителей, в том числе в приложениях Kafka Streams и коннекторов Kafka Connect, могут быть настроены следующие параметры, определяющие поведение системы при различных между предыдущими версиями и изменениями схемами:

- `auto.register.schemas` — определяет, должен ли производитель зарегистрировать схемы в Schema Registry.
- `use.latest.version` — применяется только в том случае, если `auto.register.schemas` установлено значение `false`. Если для `auto.register.schemas` установлено значение `false`, а для `use.latest.version` установлено значение `true`, то вместо получения схемы для субъекта, передаваемого клиенту для сериализации (с идентификатором) схемы с самой схемы в кэш-память производителя. При регистрации Schema Registry записывает новую последнюю версию схемы для субъекта в Kafka.
- `latest.compatibility.strict` — значение по умолчанию равно `true`, это проверяет только в том случае, если `use.latest.version` равно `true`. Если это свойство имеет значение `true`, во время сериализации производитель проверяет, чтобы убедиться, что последняя версия схемы совместима с схемой производителя. Если проверка не удалась, производитель не может отправлять данные. Если `latest.compatibility.strict` имеет значение `false`, то для сериализации используется последняя версия субъекта (без какой-либо проверки совместимости).

Подробнее о механизме чтения данных можно прочитать в статье Handling differences between preregistered and client-derived schemas.

Потребители

Ниже на рисунке показано, как работают потребители Kafka при десериализации и чтении сообщений с использованием Schema Registry.



Десериализация и чтение сообщений

Для десериализации сообщения в Kafka и экземпляру производителя подключается [десериализатор], предоставленный Schema Registry, тип которого определяет конфигурация потребителя. Десериализатор взаимодействует с сервером Schema Registry по HTTP-протоколу.

Потребитель получает данные из кластера Kafka (сериализованное сообщение) и передает их десериализатору. Десериализатор проверяет магическое magic byte и извлекает ID схемы из сообщения.

Затем десериализатор отправляет идентификатор схемы и передает, существует ли соответствующая схема в кэш-памяти потребителя. Если десериализатор не находит ID схемы, он отправляет идентификатор схемы в Schema Registry и получает схему на основе идентификатора схемы. Как только схема готова, десериализатор приступает к десериализации.

Если сериализованная данные не соответствуют полученной схеме, генерируется исключение, указывающее несоответствие между схемой и возможной ошибкой.

Для потребителей при десериализации может значение параметр `use.latest.version`, определяющий поведение системы при различении между предыдущими и зарегистрированными в клиентскими схемами.

Schema Registry в ADS

Подключение

После [обращения и установки] сервиса **Schema Registry** в составе кластера ADS вы можете подключиться к сервису Schema Registry, используя URL в формате `http://<schema-registry>:<schema-registry-port>` — указанное хоста и порта, примерами на странице информации о сервисе в интерфейсе ADCM.

Информация о сервисе Schema Registry в интерфейсе ADCM

Kafka

В ADS сервис **Schema Registry** может быть установлен только после установки сервиса Kafka. После установки Schema Registry в конфигурационном файле /etc/schema-registry/schema-registry.properties автоматически устанавливается параметр `kafkastore.bootstrap.servers` для связи с брокером Kafka (соответствующего службного) точка `listeners`.

Для тестовых целей могут быть использованы скрипты, предназначенные для работы Kafka с сериализацией, например, kafka-avro-console-producer. После установки Schema Registry эти скрипты должны быть развернуты в точке /usr/lib/schema-registry/bin/.

Ниже примеры: пример запуска производителя с указанием схемы сообщения и строкой сообщения, использующей AvroConverter в качестве параметра `key.converter` и `value.converter`, а также для параметра `key.converter.schema.registry.url` и `value.converter.schema.registry.url` указывается URL сервера Schema Registry.

Запись сообщения:

```
$ sudo /usr/lib/schema-registry/bin/kafka-avro-console-producer --bootstrap-server localhost:9092 --property schema.registry.url=http://localhost:8081 --topic topic-avro --property value.schema='{"type":"record","name":"message","fields":[{"name":"str","type":"string"},{"name":"amount","type":"double"}]}'
```

Для записи сообщения не вышедшего в: как в работе с обычными скриптами Kafka:

```
{ "id":"1000", "amount":500 }
```

Чтение сообщений из топика:

```
$ sudo /usr/lib/schema-registry/bin/kafka-avro-console-consumer --bootstrap-server localhost:9092 --from-beginning --topic topic-avro --property schema.registry.url=http://localhost:8081
```

ksqlDB

Сервис **Schema Registry** автоматически устанавливается [ksqlDB] вместе с сервисом ksqlDB. Никаких дополнительных настроек для работы с потоками и таблицами в ksqlDB. После установки Schema Registry `ksql.schema.registry.url` = `schema.registry.url`, отвечающие за взаимодействие ksqlDB и Schema Registry, можно установить на странице ksqlDB (в конфигурационных файлах /etc/ksqldb/connect.properties).

Kafka Connect

Сервис **Schema Registry** устанавливается в ADS одновременно с сервисом **Kafka Connect**, предоставляет возможность преобразования данных из внутренних типов данных, используемых Kafka Connect, в формат данных в Kafka с использованием AVRO.

Для работы с данными в формате AVRO в конфигурационных файлах Kafka Connect необходимо установить специальный AvroConverter в качестве параметра `key.converter` и/или `value.converter`, а также для параметра `key.converter.schema.registry.url` или `value.converter.schema.registry.url` указывается URL сервера Schema Registry.

Kafka REST Proxy

Сервис **Schema Registry** устанавливается в ADS одновременно с сервисом **Kafka REST Proxy**, позволяет выполнять сериализацию и десериализацию данных при работе с помощью Kafka по интерфейсу RESTful. После установки обоих сервисов параметр `schema.registry.url` автоматически устанавливается в конфигурационных файлах Kafka REST Proxy /etc/kafka-rest/kafka-rest.properties.

Используя встроенные типы схемы AVRO, PROTOBUF и JSON, вы можете напрямую встроить данные в кривом формате вместе со схемой (или идентификатором схемы) в запрос, определив соответствующий тип контента в пользовательском интерфейсе.

```
$ curl -X POST -H "Content-Type: application/vnd.kafka.avro.v2+json" \
    -H "Accept: application/vnd.kafka.avro.v2+json" \
    --data '{"value_schema": "{\"type\": \"record\", \"name\": \"User\", \"fields\": [{\"name\": \"name\", \"type\": \"string\"}]}", "records": [{"value": {"name": "testUser"}}]}' \
    "http://localhost:8082/topics/avrotest"
```

NiFi

Некоторые службы контролеров NiFi, например, ConfluentSchemaRegistry, предоставляют доступ к схемам, хранящимся на сервере Schema Registry, указанном в параметрах службы.

Настройка Schema Registry

Настройка параметров Schema Registry в интерфейсе ADCM выполняется на странице «конфигурация» сервиса **Schema Registry**

Для настройки параметров конфигурационного файла /etc/schema-registry/schema-registry.properties перейдите в дополнение состояние переключатель **Show advanced**, раскройте цвет **schema-registry.properties** и введите новые значения для параметров. Для изменения параметров Schema Registry, отсутствующих в интерфейсе ADCM, используйте поле **Add key:value**. Выберите **Add property** и введите имя/значение параметра и его значение.

После изменения параметров при помощи интерфейса ADCM перезапустите сервис **Schema Registry**. Для этого примените действие Restart с кнопки на кнопку ⋮ в столбце **Actions**.

Обнаружение сервисов в ZooKeeper

Ольга Семме

Содержание

Назад к содержанию

Обзор

Регистрация сервиса (Service Registration)

Обнаружение сервиса (Service Discovery)

ПРИМЕЧАНИЕ

Концепции и основные компоненты ZooKeeper описаны в статье ZooKeeper.

Обзор

Service Discovery — механизм ZooKeeper, действующий как реестр, в котором отслеживаются адреса всех экземпляров. Экземпляры имеют динамически назначаемые сетевые пути.

Система Service Discovery предоставляет механизм для:

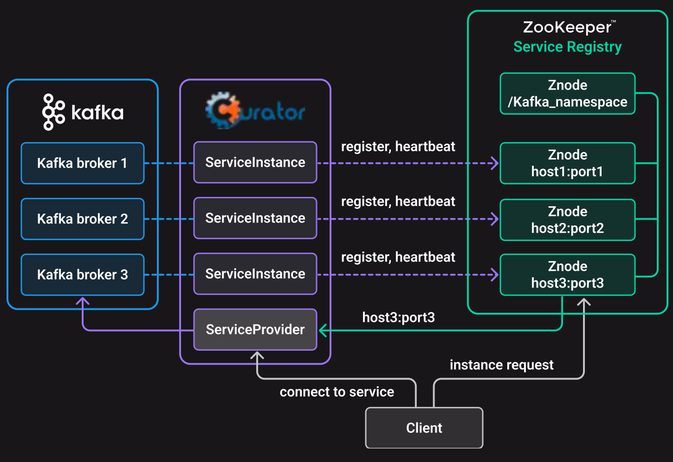
- регистрации сервисов и экземпляров сервисов;
- поиска экземпляров конкретного сервиса;
- уведомления об изменении экземпляров сервиса.

Процедура Service Discovery состоит из двух фаз:

1. Регистрация сервиса (Service Registration).
2. Обнаружение сервиса (Service Discovery).

Для ипользования механизма Service Discovery с учетом балансировки нагрузки используется пакет curator-x-discovery из состава клиентской библиотеки Java/JVM для Apache ZooKeeper Apache Curator.

Механизм Service Discovery представлен на следующей схеме.



Механизм Service Discovery в ZooKeeper

Ниже описаны стадии Service Discovery.

Регистрация сервиса (Service Registration)

Когда появляется экземпляр сервиса Kafka, он регистрируется в **Service Registry** ZooKeeper в пространстве имен (namespace) своего сервиса, добавляя свой эфемерный znode. Znode хранит `host:port` сервера в качестве своих данных. Экземпляр сервиса отправляет запросы heartbeat для подтверждения своего существования (регистрации).

Service Registry — компонент, содержащий базу данных всех доступных экземпляров сервисов. Он хранит сведения о доступных в настоящее время экземплярах каждого сервиса и их сетевые данные для установления соединения.

Service Registry отслеживает изменения в запущенных экземплярах, опрашивая среду развертывания или подписываясь на события. Когда Service Registry обнаруживает новый доступный экземпляр сервиса, он записывает его в свою базу данных. Service Registry также отменяет регистрацию отказавших (отключенных) экземпляров сервиса.

Apache Curator представляет экземпляр сервиса как класс `ServiceInstance`. `ServiceInstances` имеют имя, идентификатор, адрес, порт и/или порт ssl, а также необязательную полезную нагрузку (определяемую пользователем). `ServiceInstances` сериализуются и хранятся в ZooKeeper следующим образом:

```
base path
       |_______ service A name
                  |__________ instance 1 id --> (serialized ServiceInstance)
                  |__________ instance 2 id --> (serialized ServiceInstance)
                  |__________ ...
       |_______ service B name
                  |__________ instance 1 id --> (serialized ServiceInstance)
                  |__________ instance 2 id --> (serialized ServiceInstance)
                  |__________ ...
       |_______ ...
```

Обнаружение сервиса (Service Discovery)

После регистрации экземпляров сервисов обнаружение сервиса происходит в следующей последовательности:

1. Клиент подключается к ансамблю ZooKeeper и запрашивает нужный сервис.
2. При помощи класса `ServiceProvider`, используемого в Apache Curator, ZooKeeper использует алгоритм балансировки нагрузки для выбора одного из доступных экземпляров сервиса и возвращает `host:port` для одного из зарегистрированных экземпляров.
3. Клиент подключается к экземпляру сервиса.

Выбор лидера в ZooKeeper

Д. Ольга Семина

Содержание

Назад к содержанию

- Протокол алгоритма консенсуса
  - Обмен сообщениями
  - Принципы доставки сообщений
  - Эпоха контроллера
- Выбор контроллера Kafka
- Лидер реплицированного журнала Kafka

ПРИМЕЧАНИЕ

- Концепции хранения и основные компоненты Kafka описаны в статье «Концепции хранения в Kafka».
- Концепции и основные компоненты ZooKeeper описаны в статье ZooKeeper.

ZooKeeper — централизованный сервис, используемый для хранения информации о конфигурации, обеспечения распределенной синхронизации брокеров Kafka.

Установление системы обмена сообщениями, лежащей в основе ZooKeeper, основана на принципах алгоритма консенсуса.

Алгоритм консенсуса — совокупность принципов и правил, благодаря которым все участвующие в кластере узлы (лидер) автоматически приходят к консенсусу о текущем состоянии сети.

ZooKeeper реализует один из протоколов алгоритма консенсуса – ZAB (ZooKeeper Atomic BroadCast).

Протокол алгоритма консенсуса

Протокол ZAB гарантирует, что репликации ZooKeeper выполняются по порядку, а также отвечает за выбор ведущих узлов и восстановление любых отказавших узлов.

Ниже описаны основные положения ZAB.

Обмен сообщениями

В части обмена сообщениями ZAB использует следующие понятия:

- **Пакет (packet)** – последовательность байтов, отправляемых по каналу FIFO.
- **Сообщение (message)** – последовательность байтов, которая будет атомарно транслироваться на все серверы ZooKeeper. Сообщение помещается в предложение и согласовывается до того, как оно будет доставлено.
- **Предложение (proposal)** – единица согласования. Предложение согласовывается путем обмена пакетами при наличии кворума серверов ZooKeeper.

Ниже показана модель обмена сообщениями, основанная на согласовании предложений, принятая в ZooKeeper.

Модель обмена сообщениями

Для связи с репликами при согласовании предложений используется модель, в которой для каждой записи требуется не менее трех транзакций: предложение (propose), подтверждение (ack) и фиксация (commit).

Принципы доставки сообщений

Протокол ZAB придерживает создания каналов FIFO (в каждой сессии сервер-клиент запросы клиента полагаются в порядке их поступления) типа "точка-точка" между серверами. Для связи используется протокол TCP, обеспечивая следующие свойства:

- **Надежная доставка** – если сообщение **a** доставляется одним сервером, оно в конечном итоге будет доставлено всеми серверами.
- **Общий заказ** – если сообщение **a** доставляется до сообщения **b** одним сервером, то **a** будет доставлено до **b** всеми серверами. Если сообщения **a** и **b** доставлены, то либо **a** будет доставлено раньше **b**, либо **b** будет доставлено раньше **a**.
- **Причинный порядок** – если сообщение **a** отправляется после того, как сообщение **b** было доставлено отправителем **a**, сообщение **a** должно быть заказано до **b**. Если отправитель отправляет **a** после отправки **b**, **a** должно быть заказано после **b**.
- **Заказанная доставка** – данные доставляются в том же порядке, в котором они были отправлены, и сообщения **a** доставляются только после того, как все сообщения, отправленные до **a**, были доставлены. Сообщения могут быть доставлены только тогда, когда все сообщения, которые они должны заказать до **a**, будут получены.
- **Нет сообщения после закрытия** – как только канал FIFO закрыт, от него не будет приниматься никакие сообщения.

Использование **total-order** обеспечивает достижение консенсуса при наличии сбоев.

Каждый кластер имеет один лидер (**Leader**), а остальные узлы являются последователями (**Follower**).

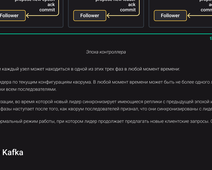
Для гарантии общего порядка предложений используется идентификатор транзакции ZooKeeper - **zxid**.

Zxid состоит из двух частей: номера эпохи и счетчика транзакции.

Zxid является 64-битным числом – старшие 32 бита для номера эпохи и младшие 32 бита для счетчика транзакции.

Новый номер эпохи присваивается лидеру, когда он получает кворум голосов от каждого лидера, претендовавшего своей уникальной zxid.

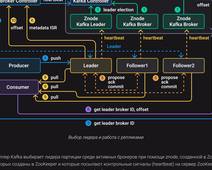
Ниже приведена общая последовательность выбора лидера.



Выбор лидера в ZooKeeper

1. От сервера поступает предложение (propose) новой эпохи и каждому предложению присваивается свой zxid.
2. Каждый узел подтверждает (ack) предложение только в том случае, если он не знает ни одного другого лидера с более высоким номером эпохи в составе zxid, или, если эпоха такая же, с более высоким счетчиком транзакции. Прежде чем лидер будет выбран, он должен собрать голоса от кворума узлов. Каждому для определения лидера должны иметь уникальный (**id**,**zxid**), где (**id**) — количество серверов, составляющих кворум ZooKeeper.
3. После подтверждения кворума новый лидер совершает (commit) создание новой эпохи и устанавливает следующий zxid для использования.
4. Последователь зафиксирует становление новой эпохи.

Ниже описаны все фазы эпохи контроллера.



Эпоха контроллера

Каждая эпоха состоит из трех фаз, и каждый узел может находиться в одной из этих трех фаз в любой момент времени:

- **Leader election** – фаза выбора лидера по текущим конфигурациям кворума. В любой момент времени может быть не более одного лидера. Окончание фазы наступает после подтверждения новой эпохи всеми последователями.
- **Synchronization** – фаза синхронизации, во время которой новый лидер синхронизирует имеющиеся реплики с предыдущей эпохой и со всеми последователями перед началом трансляции. Последователи получают после того, как лидер установил последователей признак, что они синхронизированы с лидером.
- **Broadcast** – фаза трансляции, нормальной режим работы, при котором лидер продолжает предлагать новые клиентские запросы. Окончание фазы наступает после сбоя лидера.

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.

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

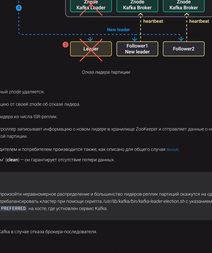
1. Каждый брокер пытается создать эфемерный znode с наименованием `/controller` (название может отличаться в ZooKeeper).
2. Первый брокер, создавший этот эфемерный znode, возьмет на себя роль контроллера, и каждый последующий запрос брокера будет получать сообщение `node already exists`.
3. После того, как контроллер установлен, ему назначается "эпоха контроллера".
4. Текущая эпоха контроллера передается по всему кластеру, и если брокер получает запрос контроллера от более старой эпохи контроллера, он игнорируется.
5. Если происходит сбой контроллера, происходит новый выбор контроллера в соответствии с принципами выбора лидера протокола ZAB, описанного выше, и создается новая эпоха контроллера, которая передается в кластере.

Лидер реплицированного журнала Kafka

Лидер Leader журнала (партиции) – брокер, который работает с клиентами. Именно лидер работает с производителями сообщений. К задаче осуществляет запросы **последователи followers** – брокеры, которые реплицируют данные лидера (партиций). Потребители могут читать сообщения как с лидера, так и с последователя.

Алгоритм распределенного консенсуса ZAB также используется для согласования между последовательностью записи и чтения сообщений и управлением репликацией партиций в Kafka для обеспечения доступности и надежности системы.

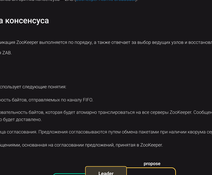
Ниже описан, как в общем случае автоматическом осуществляется выбор лидера и управление репликацией партиций в Kafka при помощи сервиса ZooKeeper.



Выбор лидера и работа с репликами

1. После создания топика контроллер Kafka выбирает лидера партиции среди активных брокеров при помощи znode, созданной в ZooKeeper. Активными считаются брокеры, эфемерные znode которых создаются в ZooKeeper и поддерживают контрольные сигналы (heartbeat) на сервис ZooKeeper для поддержания сессии. Для отслеживания состояния брокеров нода контроллера подписывается (watches) на ноды брокеров.
2. Получив О нового лидера, контроллер записывает информацию о новом лидере в хранилище ZooKeeper и отправляет данные о новом лидере 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. Последователи, которые содержат реплики всех записанных сообщений – **ISR-реплика** – являются кандидатами в лидеры при отказе действующего лидера.
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 последователя для запрашиваемой партиции, а также данные о смещении (offset), с которого следует читать.
9. Потребитель запрашивает (pull) сообщения у лидера или последователя.
10. Лидер получает данные о смещении (offset) и передает их в ZooKeeper.

Ниже описаны, как осуществляется выбор лидера в Kafka в случае отказа лидера партиции.



Отказ лидера партиции

1. При отказе лидера его эфемерный znode удаляется.
2. Контроллер получает информацию от своей znode об отказе лидера.
3. Контроллер выбирает нового лидера из числа ISR-реплик.
4. Получив ID нового лидера, контроллер записывает информацию о новом лидере в хранилище ZooKeeper и отправляет данные о новом лидере каждому брокеру, который размещает реплику этой партиции.

Далее работа с репликами, производителем и потребителем производится также, как описано для общего случая выше.

Этот выбор лидера является "честным" (**Clean**) – он гарантирует отсутствие потери данных.

ПРИМЕЧАНИЕ

После отказов брокеров может произойти неравномерное распределение и большинство лидеров ролями партиций окажутся на одном узле кластера. В этом случае можно вручную перебалансировать кластер при помощи скрипта `opt/kafka/bin/kafka-leader-election.sh` с указанием опции `--election-type` со значением `PREFERRED` на кластере, где установлен сервис Kafka.

Отказ последователя

1. Лидер ждет ответ о создании реплики (ack) от последователя по истечении заданного времени (время ответа определяется параметром Kafka-брокера `replica.lag.time.max.ms` по умолчанию 30000 мс).
2. При истечении времени лидер считает реплику несинхронизированной и удаляет ее из ISR.
3. Информация об удалении ISR поступает в ZooKeeper. Данные последователь перестает быть кандидатом в лидеры партиции.

Далее работа с репликами, производителем и потребителем производится также, как описано для общего случая выше.

ВНИМАНИЕ

Если включить параметр Kafka-брокера `unclean.leader.election.enable` , новым лидером для партиций будут создаваться не из списка ISR. При этом будут потеряны все несинхронизированные сообщения, которые были отправлены старому лидеру. В случае, когда нет ISR, хорам ставшего недоступным лидера, и позволит лидера в сеть невозможен, это единственный вариант оставить топик доступным для потребителей. Для отдельного топика это можно выполнить при помощи скрипта /usr/lib/kafka/bin/kafka-configs.sh с использованием опции `--add-config` с указанием параметра `unclean.leader.election.enable=true`

Формат сообщений в Kafka

Ольга Семме

Содержание

Назад к содержанию

Запись сообщений

Формат сообщения

Формат пакета записей

Хранение записи на брокере

В данной статье описан формат сообщений в Kafka.

Сообщение (message), запись (record) – фиксация в системе произошедшего события, в том числе состояния объекта, значения физической величины или любого другого параметра, который требует отслеживания, хранения или передачи в другую систему.

В процессе настройки компонентов, принимающих участие в записи и чтении сообщений, имеет значение понимание формата сообщений и настройка связанных с ним параметров.

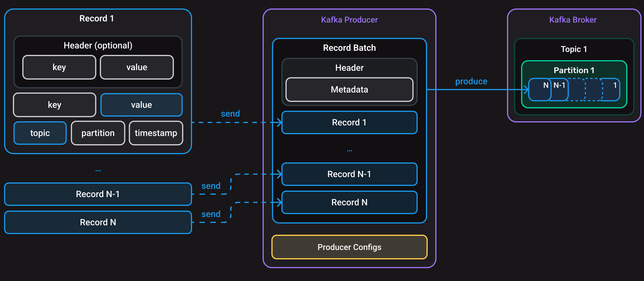
Запись сообщений

Сообщения в Kafka сохраняются (буферизуются) в **пакеты записей (record batch)**, а затем с учетом конфигураций производителя отправляются в брокер Kafka. Пакет может содержать от одного до нескольких сообщений. При этом сообщения одного пакета записываются в одну и ту же партицию.

Регулирование пакетной передачи сообщений выполняется при помощи параметров:

- max.in.flight.requests.per.connection – максимальное количество неподтвержденных запросов (количество пакетов) на запись, которые могут быть отправлены одновременно (по умолчанию значение `5`);
- linger.ms – настройка времени искусственной задержки отправления пакетов для объединения записей в пакет (по умолчанию значение `0`);
- batch.size – настройка верхней границы размера пакета в байтах (по умолчанию значение `16384`).

Принцип пакетной обработки сообщений в Kafka показан на рисунке ниже.



Запись сообщений в Kafka

Формат сообщения

Основой сообщения Kafka является пара **ключ/значение (key/value)**. И ключ, и значение представляют собой массив байтов переменной длины. Значение может представлять собой любой массив данных, в том числе и определенное количество пар ключ/значение. Ключ может быть равен `null` или не указываться вообще.

И ключ, и значение сообщения могут быть сериализованы в одном из форматов: json, avro, protobuf. Сериализация выполняется при запуске экземпляра производителя.

При создании сообщения вместе с парой ключ/значение указываются следующие параметры:

- наименование **топика (topic)**, куда должно быть записано сообщение;
- номер **партиции (partition)**, в которую должно быть записано сообщение (может быть не указан);
- **временная метка (timestamp)**, указанная пользователем (может отсутствовать).

Исходя из указанных ключа и номера партиции, при создании сообщения определяется одна из стратегий записи сообщения в определенную партицию топика:

- запись сообщения в указанную партицию, если данная партиция существует;
- запись сообщений с одинаковым ключом в одну партицию, которая вычисляется при помощи хеш-функции ключа (если не указан номер партиции и присутствует ключ);
- запись сообщения в партицию, выбранную **циклическим способом (round-robin)** (ни партиция, ни ключ не указаны).

Если при создании сообщения пользователь не установил временную метку, сообщение будет отмечено текущим временем.

При помощи заголовка записи (**header**) существует возможность добавить метаданные для каждого сообщения. Заголовок представляет собой пару ключ/значение, где ключ – всегда закодированная строка, а значение может быть любым массивом данных, в том числе и определенное количество пар ключ/значение.

Создание записи с указанием необходимых параметров Kafka выполняется с использованием класса ProducerRecord.

Ниже приведен дисковый формат записи и заголовка записи с указанием используемых типов данных для всех значений.

Дисковый формат записи

```
length: varint
attributes: int8
    bit 0~7: unused
timestampDelta: varlong
offsetDelta: varint
keyLength: varint
key: byte[]
valueLen: varint
value: byte[]
Headers => [Header]
```

Дисковый формат заголовка

```
headerKeyLength: varint
headerKey: String
headerValueLength: varint
Value: byte[]
```

Формат пакета записей

После создания записи запускается экземпляр KafkaProducer, который выполняет:

1. Сериализацию записей в соответствии с установленными параметрами сериализации key.serializer и value.serializer.
2. Партиционирование записей в соответствии с выбранной стратегией.
3. Буферизацию записей и сортировку по пакетам.
4. Выполнение необходимого сжатия сообщений внутри пакета в соответствии с установленными параметрами сжатия compression.type.

Пакет записей содержит от одной до нескольких записей и **заголовок (header)** — раздел метаданных пакета записей. Заголовок всегда содержит 61 байт. Заголовок содержит данные о типе используемого сжатия, метку создания пакета, количество записей и другие метаданные.

Ниже приведен дисковый формат заголовка пакета записей с указанием используемых типов данных для всех значений.

Дисковый формат заголовка пакета записей

```
baseOffset: int64
batchLength: int32
partitionLeaderEpoch: int32
magic: int8 (current magic value is 2)
crc: uint32
attributes: int16
    bit 0~2:
        0: no compression
        1: gzip
        2: snappy
        3: lz4
        4: zstd
    bit 3: timestampType
    bit 4: isTransactional (0 means not transactional)
    bit 5: isControlBatch (0 means not a control batch)
    bit 6: hasDeleteHorizonMs (0 means baseTimestamp is not set as the delete horizon for compaction)
    bit 7~15: unused
lastOffsetDelta: int32
baseTimestamp: int64
maxTimestamp: int64
producerId: int64
producerEpoch: int16
baseSequence: int32
records: [Record]
```

Хранение записи на брокере

После записи на брокер Kafka сообщение сохраняется со всеми метаданными, которые были записаны в процессе записи сообщения, при этом:

- Сообщение на брокере имеет временную метку исходя из настроенного параметра брокера log.message.timestamp.type или параметра топика message.timestamp.type.
- Сжатие записанных сообщений на брокере выполняется исходя из настроенного параметра брокера compression.type или параметра топика compression.type. Сообщение в сжатом виде сохраняется в брокере и распаковывается только у потребителя.
- Сообщение на брокере сохраняется в сериализованном виде исходя из параметров сериализации установленных производителем.

Класс RecordMetadata возвращает метаданные записи, успешно записанной на брокере.

Объекты NiFi

Ольга Семин

Содержание

Назад к содержанию

- FlowFile
- Процессор
- Соединение
- Репозитории
  - FlowFile Repository
  - Content Repository
  - Provenance Repository
- Обработка FlowFile
- Process Group
- Flow Controller
- Настройка NiFi

В данной статье описана архитектура основных объектов NiFi и их взаимодействие.

FlowFile

FlowFile – компонент NiFi, представляющий собой пакет данных. Это основная единица обработки и передачи информации в NiFi.

Каждый FlowFile состоит из двух частей:

- **содержимое (content)** – массив байтов, который должен быть преобразован, извлечен или передан при помощи инструментов NiFi. Само содержимое хранится в Content Repository, и FlowFile отслеживает путь к месту его хранения.
- **атрибуты (attributes)** – пары ключ/значение, которые содержат сведения о содержимом и другие данные или параметры, необходимые для правильной обработки.

Процессор

Процессор (Processor) – компонент NiFi, осуществляющий обработку FlowFiles. Процессор состоит из логического блока, реализованного на Java в соответствии с заданным назначением процессора, и интерфейса, при помощи которого возможно:

- выполнять настройку конфигурационных параметров процессора, указывающих, как будут обработаны данные, и определяющих связь с внешними процессами;
- управлять процессором;
- осуществлять диагностику состояния процессора.

Соединение

Соединение (connection) – компонент NiFi, соединяющий процессоры для передачи FlowFiles между ними. В NiFi соединение действует как буфер или очередь FlowFiles. После обработки FlowFiles одним процессором они накапливаются в очереди соединения, пока не будут приняты на обработку другим процессором. Очереди позволяют процессорам взаимодействовать с разной скоростью.

Соединение NiFi состоит из очереди FlowFiles и интерфейса, при помощи которого возможно:

- выполнять настройку параметров соединения, которые определяют, как будут обрабатываться очереди, в том числе:
  - настройку применения backpressure;
  - настройку приоритетности FlowFiles;
- управлять соединением;
- осуществлять диагностику очереди.

Репозитории

FlowFile Repository

Репозиторий **FlowFile Repository** сохраняет данные каждого FlowFile в формате **журнала опережающей записи (Write Ahead Log, WAL)**. Каждое изменение FlowFile является транзакционной операцией, что гарантирует возможность восстановления без потери данных при сбоях.

Информация о хранении FlowFile в репозитории доступна при просмотре данных FlowFile.

Система периодически создает моментальный снимок (снепшот) текущего состояния FlowFile (атрибуты и путь к содержимому на диске) и записывает данные снимка в репозиторий в виде файла с расширением *.partial*. Эти файлы являются промежуточными записями транзакций обработки FlowFile. В момент наступления **контрольной точки (checkpoint)** последний снепшот *.partial* записывается в *snapshot* (интервал между контрольными точками управляется параметром `nifi.flowfile.repository.checkpoint.interval`). В случае сбоя файлы *.partial*, записанные после последнего файла *snapshot*, удаляются, а последний файл *snapshot* считается точкой восстановления для узла. Если файла *snapshot* не существует, последний *.partial* переименовывается в *snapshot*.

Для увеличения быстродействия обрабатываемые FlowFile могут храниться в памяти JVM, что может привести к потере данных при сбоях. С учетом использования WAL риски могут быть минимизированы при соответствующей требованиям проекта настройке параметров.

Процесс работы NiFi с памятью JVM (хранение данных и подкачка файлов при обработке) может быть отрегулирован при помощи параметров FlowFile Repository:

- `nifi.flowfile.repository.implementation` – определяет место хранения обрабатываемых файлов;
- `nifi.flowfile.repository.always.sync` – определяет, нужно ли синхронизировать любое изменение в репозитории с диском;
- `nifi.queue.swap.threshold` – порог очереди (количество FlowFile), при котором NiFi начинает подкачку (swap) информации FlowFile на диск.

Content Repository

Содержимое каждого FlowFile хранится в **Content Repository**. Репозиторий состоит из набора файлов на диске, каждый из которых имеет свой **идентификатор (identifier)**. Содержимое каждого FlowFile хранится в файле под определенным **смещением (offset)**.

Ниже приведена иерархия хранения данных в Content Repository:

```
base path
      |__Container 0
                  |__Section 1
                              |__identifier
                                          |__offset
                  |__Section 2
                              |__identifier
                                          |__offset
                  |_________ ...
      |__Container 1
                  |__Section 1
                  |__Section 2
                  |__ ...
      |__...
```

Этот репозиторий использует парадигмы **неизменяемости (immutability)** и **копирования (copy-on-write)**. Это значит, что данные, записанные в репозиторий, не меняются и не меняют места размещения на диске.

Местонахождение содержимого отслеживает объекты Java **Resource Claims**, которые указывают путь к содержимому на диске (Container → Section → identifier → offset) и передают эти данные в качестве объекта **Content Claim**, который является частью метаданных FlowFile при хранении в FlowFile Repository.

Resource Claim отслеживает один и тот же путь для нескольких FlowFile, для которых содержимое является одинаковым. Если процессор меняет содержимое, новые данные записываются в новый фрагмент на диске и Resource Claim начинает отслеживать новый путь хранения данных для передачи в Content Claim.

Если ссылки на содержимое, созданные объектами Resource Claim, не используются ни в одном Content Claim, данное содержимое будет удалено или архивировано (архивирование включается параметром `nifi.content.repository.archive.enabled`, а время хранения архивных данных параметром `nifi.content.repository.archive.max.retention.period`).

Отслеживание неиспользующихся ссылок происходит каждый раз в момент достижения контрольной точки в **FlowFile Repository**.

Provenance Repository

Репозиторий **Provenance Repository** сохраняет снепшоты данных каждого FlowFile каждый раз после того, как FlowFile был изменен процессором. Это позволяет проследить хронологию изменения FlowFile.

Такой снимок в NiFi называется **событием происхождения (Provenance Event)**.

Все события происхождения отображаются на странице Data Provenance глобального меню пользовательского интерфейса.

Старые версии содержимого FlowFile могут быть удалены до того, как будут удалены события происхождения, ссылающиеся на них. При этом пользователю будут доступны данные FlowFile для анализа происхождения данных.

Период хранения информации о происхождении данных регулируется параметром `nifi.provenance.repository.max.storage.time`.

Обработка FlowFile

Ниже на рисунке показано, как происходит обработка и изменение данных в NiFi.

Объекты NiFi

Ниже описана последовательность обработки FlowFile:

1. Снепшоты данных FlowFile с определенной периодичностью записываются в FlowFile Repository (`F1–F4`).
2. Если логика процессора предполагает изменение содержимого, FlowFile запрашивает содержимое (`C1–C4`) из Content Repository в соответствии с Content Claim и считывает его в память JVM.
3. Измененное содержимое (`N1–Nn`) записывается в новый фрагмент диска.
4. Объект Resource Claim начинает отслеживать новый путь к содержимому и обновляет данные в Content Claim.
5. В Provenance Repository записывается новый снепшот метаданных и атрибутов FlowFile (событие происхождения).
6. FlowFile поступает в очередь соединения для передачи и обработку следующему процессору. В FlowFile Repository записывается новый снепшот данных FlowFile.

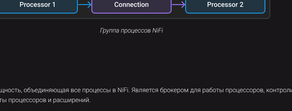
Если во время выполнения одного из этапов возникает сбой, система вернется к состоянию, соответствующему последнему файлу *snapshot* в FlowFile Repository. При этом новые данные, записанные на диск, будут удалены, так как ссылка на них не успеет обновиться, а доступ к старым данным сохранится благодаря копированию при записи и неизменяемости.

Process Group

Группа процессов (Process Group) – компонент NiFi, позволяющий объединять функциональные возможности разных процессоров и других элементов NiFi в один компонент, являющийся по сути новым процессором. Группа процессов должна содержать **входной порт (Input Port)**, **выходной порт (Output Port)** и может содержать любое количество элементов NiFi и соединений между ними.

Группа процессов имеет интерфейс, при помощи которого возможно:

- выполнять настройку конфигурационных параметров группы процессов;
- управлять группой процессов;
- осуществлять диагностику состояния группы процессов.



Группа процессов NiFi

Flow Controller

Контроллер потока (Flow Controller) – сущность, объединяющая все процессы в NiFi. Является брокером для работы процессоров, контролирует работу расширений NiFi и планирует выделение ресурсов для работы процессоров и расширений.

Контроллер потока позволяет добавлять службы контроллера.

Службы контроллеров (Controller Service) – службы, при помощи которых могут быть определены общие настройки подключения к внешним системам и распространены на множество процессоров или групп процессов.

Настройка контроллера потока и сервисов контроллера выполняется на странице Controller Settings глобального меню пользовательского интерфейса.

Настройка NiFi

После добавления и установки сервиса **NiFi** в составе кластера ADS настройка параметров выполняется на странице конфигурирования сервиса **NiFi** в интерфейсе ADCM.

Для настройки параметров работы сервиса NiFi перейдите в активное состояние переключатель ***Show advanced***, раскройте узел ***nifi.properties*** и введите новые значения для параметров. Для изменения параметров NiFi, отсутствующих в интерфейсе ADCM, используйте поле ***Add key:value*** в узле ***nifi.properties***. Выберите ***Add property*** и введите наименование параметра и его значение.

Типы процессоров NiFi

Denis Chmel

Содержание

Назад к содержанию

Data transformation
Routing and mediation
Database access
Attribute extraction
System interaction
Data ingestion
Data egress/Sending data
Splitting and aggregation
HTTP
Amazon Web Services

В данной статье описаны существующие типы процессоров NiFi.

ПРИМЕЧАНИЕ

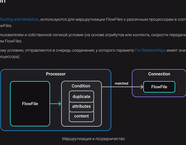
Для полученя информации об архитектуре основных объектов NiFi и их взаимодействии вы можете обратиться к статье Объекты NiFi.

Data transformation

Процессоры, принадлежащие типу Data Transformation, меняют содержимое FlowFile.

Процессоры, которые относятся к Content Repository, преобразуют данные в соответствии с собственной логикой, записывают новые данные в Content Repository и сохраняют ссылку на изменённое данное в FlowFile.

FlowFiles, содержимое которых удалось преобразовать, отправляются в очередь соединения, у которого параметр For Relationships имеет значение `success`.



Преобразование данных

Примеры процессоров и их использование:

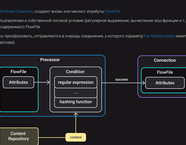
- замена всего или части текста при помощи регулярного выражения – ReplaceText;
- сжатие или распаковка содержимого – CompressContent;
- преобразование символов из одного набора символов в другой – ConvertCharacterSet.

Routing and mediation

Процессоры, принадлежащие типу Routing and Mediation, используются для маршрутизации FlowFiles в различными процессорам в соответствии с информацией в атрибутах или содержимом этих FlowFiles.

Процессоры используют заданную пользователем и собственной логикой условие (на основе атрибутов или контента, скорости передачи данных, отслеживания дубликатов и т.д.) для управления перемещением FlowFiles.

FlowFiles, соответствующие заданному условию, отправляются в очередь соединения, у которого параметр For Relationship имеет значение `success`, `matched`, `duplicate` – в зависимости от процессора.



Маршрутизация и посредничество

Примеры процессоров и их использование:

- регулирование пропускной способности FlowFiles на основе настраиваемой скорости – ControlRate;
- отслеживание дубликатов FlowFiles на основе контента, определяемых пользователем – DetectDuplicate;
- маршрутизация FlowFiles на основе их атрибутов с использованием языка выражений NiFi – RouteOnAttribute.

Database access

Процессоры, принадлежащие типу Database Access, применяются для работы с базами данных.

Процессоры выполняют SQL-запрос, где этим в зависимости от собственной логики может использовать содержимое входящего FlowFile в качестве содержания запроса или записывают результат запроса в качестве содержимого исходного FlowFile.

Такие некоторые процессоры могут подготовливать запросы, хранящиеся на и других форматов (ConvertJSONToSQL).

FlowFiles при успешном выполнении запроса отправляются в очередь соединения, у которого параметр For Relationships имеет значение `success`.

Доступ к базе данных

Примеры процессоров и их использование:

- преобразование документа JSON в команду SQL `INSERT` или `UPDATE`, которую затем можно передать процессору PutSQL – ConvertJSONToSQL;
- обновление базы данных с использованием в команде содержимого FlowFile – PutSQL;
- выполнение команды SQL `SELECT` с записью результатов в содержимое FlowFile в формате Avro – ExecuteSQL.

Attribute extraction

Процессоры, принадлежащие типу Attribute Extraction, создают вместо или меняют атрибуты FlowFile.

Процессоры используют заданное пользователем и собственной логикой условие (регулярное выражение, вычисление язык функции и т.д.), чтобы изменить уже имеющиеся атрибуты или извлечь атрибуты из содержимого FlowFile.

FlowFiles, атрибуты которых удалось преобразовать, отправляются в очередь соединения, у которого параметр For Relationships имеет значение `success` или `matched` – в зависимости от процессора.

Извлечение атрибутов

Примеры процессоров и их использование:

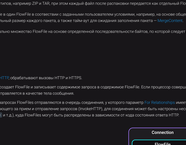
- извлечение атрибутов из текста содержимого FlowFile в соответствии с регулярными выражениями, заданными в пользовательских свойствах процессора – ExtractText;
- обновление атрибутов FlowFile с помощью языка выражений NiFi или на основе регулярных выражений, заданных пользователем – UpdateAttribute;
- вычисление криптографического хеш-значения для содержимого FlowFile с использованием заданного алгоритма и запись значения в выходной атрибут – CryptographicHashContent.

System interaction

Процессоры, принадлежащие типу System Interaction, запускают процессы в операционной системе или скрипты в различных средах разработки.

Процессоры запускают команду операционной системы или скрипт с использованием содержимого входящего FlowFile или без него. При этом процессор создаёт или изменяет FlowFile, в содержимое которого записывается результат отработки команды или скрипта.

При успешной обработке команда создаёт FlowFile, отправляемый в очередь соединения, у которого параметр For Relationships имеет значение `success`.



Взаимодействие с системой

Примеры процессоров и их использование:

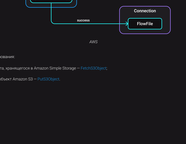
- запуск заданной пользователем команды операционной системы и создание FlowFile в результате отработки команды – ExecuteProcess;
- запуск заданной пользователем команды операционной системы с использованием содержимого входящего FlowFile как stdin и записью stdout в исходящий FlowFile – ExecuteStreamCommand.

Data ingestion

Процессоры, принадлежащие типу Data Ingestion, принимают данные в поток NiFi.

Процессоры создают новый FlowFile, при этом загружают содержимое исходного файла (или другого источника) в Content Repository и создают атрибуты FlowFile. Как правило, такие процессоры являются отправной точкой для потоков данных в Apache NiFi.

При успешной записи содержимого и создании атрибутов FlowFiles отправляются в очередь соединения, у которого параметр For Relationships имеет значение `success`.



Прием данных

Примеры процессоров и их использование:

- подписка содержимого файла с локального диска (или сетевого диска) в качестве содержимого создаваемого FlowFile и последующее удаление исходного файла – GetFile;
- подписка на один или несколько топиков Apache Kafka с созданием FlowFile, в качестве содержимого которого сохраняется сообщение Apache Kafka – ConsumeKafka.

Data egress/Sending data

Процессоры, принадлежащие типу Data Egress/Sending Data, отправляют данные на сервер назначения.

Процессоры передают содержимое FlowFile в точку назначения, а также отправляют атрибуты в качестве дополнительных заголовков, если это указано пользователем в настройках процессора. Как правило, такие процессоры являются конечной точкой потока данных в Apache NiFi.

При успешной отправке данных в создаётся FlowFile без данных.

Выход данных

Примеры процессоров и их использование:

- отправление содержимого входящего FlowFile электронным письмом получателю – PutEmail;
- отправление содержимого FlowFile в виде сообщения в Apache Kafka – PublishKafka;
- запись содержимого FlowFile в файл локальной системы – PutFile.

Splitting and aggregation

Процессоры, принадлежащие типу Splitting and Aggregation, разделяют или объединяют содержимое FlowFile.

При разделении процессор создаёт новые FlowFiles, в содержимое которых записываются данные, полученные в результате разделения. Разделение происходит в соответствии с заданным пользователем условием и собственной логикой процессора. Атрибуты для каждого нового FlowFile обычно являются дочерними, относительно полученный в результате деления имеет содержимое – размер, идентификатор, количество строк и т.д.

При агрегировании процессор создаёт новый FlowFile, содержимое которого объединяет содержимое всех входящих файлов. Атрибуты для нового FlowFile являются дочерними, относительно полученный в результате объединения имеет файла, количество объединяемых FlowFiles, время существования таких до агрегирования и т.д.

Процессоры создают сообщение новых FlowFiles в Content Repository и сохраняют ссылку на них в каждом FlowFile.

При разделении или агрегировании содержимого новые FlowFiles отправляются в очередь соединения, у которого параметр For Relationships имеет значение `success`, `splits`, `segments`, `merged` – в зависимости от процессора.

Разделение

Примеры процессоров и их использование:

- Распаковка архивов различных типов, например, ZIP и TAR, при этом каждый файл после распаковки передается как отдельный FlowFile – UnpackContent;
- объединение множества FlowFile в один FlowFile в соответствии с заданными пользователем условиями, например, на основе атрибута. При этом может быть задан минимальный и максимальный размер пакета, а также тайм-аут для ожидания достаточного пакета – MergeContent;
- Разделение FlowFile на потенциально множество FlowFile на основе определенного пользовательского battne, по которой следует разделить содержимое – SplitContent.

HTTP

Процессоры, принадлежащие типу HTTP, обрабатывают вызовы HTTP и HTTPS.

При вызывающие запросов процессор создает FlowFile и записывает содержимое запроса в содержимое FlowFile. Если процессор совершает исходящий запрос (PUT, POST или PATCH), содержимое FlowFile отправляется в качестве тела сообщения.

Успешно созданные при входящих запросов FlowFiles отправляются в очередь соединения, у которого параметр For Relationships имеет значение `success`. Для универсального процессора, отвечающего на запросы и отправляющего запросы (InvokeHTTP), для соединения может быть настроены несколько значений **For Relationships** `original`, `failure`, `response` и т.д, куда FlowFiles могут быть распределены в зависимости от кода состояния ответа HTTP.

HTTP-запросы

Примеры процессоров и их использование:

- запуск HTTP- (или HTTPS-) сервера и прослушивание входящих соединений – ListenHTTP;
- взаимодействие с настраиваемой конечной точкой HTTP-с использованием методов GET, при котором тело ответа становится содержимым создаваемого FlowFile – InvokeHTTP.

Amazon Web Services

Процессоры, принадлежащие типу Amazon Web Services, отвечают за взаимодействие с системой веб-сервисов Amazon.

Процессоры совершают действия в соответствии с собственной логикой – извлекают содержимое объектов или сообщений AWS, отправляют содержимое FlowFile в качестве содержимого объекта AWS или уведомлений. При создании данных из AWS в содержимое FlowFile в качестве атрибутов сохраняются параметры, специфичные данным идентификаторы, мер-сервиса и т.д.

При успешном выполнении запроса FlowFiles отправляются в очередь соединения, у которого параметр For Relationships имеет значение `success`.

AWS

Примеры процессоров и их использование:

- извлечение содержимого объекта, хранящегося в Amazon Simple Storage – FetchS3Object;
- запись содержимого FlowFile в объект Amazon S3 – PutS3Object.

Revision 2025

Table of contents

1. Abstract
2. Abstract
3. Abstract
4. Abstract
5. Abstract
6. Abstract
7. Abstract
8. Abstract
9. Abstract
10. Abstract
11. Abstract
12. Abstract
13. Abstract
14. Abstract
15. Abstract
16. Abstract
17. Abstract
18. Abstract
19. Abstract
20. Abstract
21. Abstract
22. Abstract
23. Abstract
24. Abstract
25. Abstract
26. Abstract
27. Abstract
28. Abstract
29. Abstract
30. Abstract
31. Abstract
32. Abstract
33. Abstract
34. Abstract
35. Abstract
36. Abstract
37. Abstract
38. Abstract
39. Abstract
40. Abstract
41. Abstract
42. Abstract
43. Abstract
44. Abstract
45. Abstract
46. Abstract
47. Abstract
48. Abstract
49. Abstract
50. Abstract
51. Abstract
52. Abstract
53. Abstract
54. Abstract
55. Abstract
56. Abstract
57. Abstract
58. Abstract
59. Abstract
60. Abstract
61. Abstract
62. Abstract
63. Abstract
64. Abstract
65. Abstract
66. Abstract
67. Abstract
68. Abstract
69. Abstract
70. Abstract
71. Abstract
72. Abstract
73. Abstract
74. Abstract
75. Abstract
76. Abstract
77. Abstract
78. Abstract
79. Abstract
80. Abstract
81. Abstract
82. Abstract
83. Abstract
84. Abstract
85. Abstract
86. Abstract
87. Abstract
88. Abstract
89. Abstract
90. Abstract
91. Abstract
92. Abstract
93. Abstract
94. Abstract
95. Abstract
96. Abstract
97. Abstract
98. Abstract
99. Abstract
100. Abstract
101. Abstract
102. Abstract
103. Abstract
104. Abstract
105. Abstract
106. Abstract
107. Abstract
108. Abstract
109. Abstract
110. Abstract
111. Abstract
112. Abstract
113. Abstract
114. Abstract
115. Abstract
116. Abstract
117. Abstract
118. Abstract
119. Abstract
120. Abstract
121. Abstract
122. Abstract
123. Abstract
124. Abstract
125. Abstract
126. Abstract
127. Abstract
128. Abstract
129. Abstract
130. Abstract
131. Abstract
132. Abstract
133. Abstract
134. Abstract
135. Abstract
136. Abstract
137. Abstract
138. Abstract
139. Abstract
140. Abstract
141. Abstract
142. Abstract
143. Abstract
144. Abstract
145. Abstract
146. Abstract
147. Abstract
148. Abstract
149. Abstract
150. Abstract
151. Abstract
152. Abstract
153. Abstract
154. Abstract
155. Abstract
156. Abstract
157. Abstract
158. Abstract
159. Abstract
160. Abstract
161. Abstract
162. Abstract
163. Abstract
164. Abstract
165. Abstract
166. Abstract
167. Abstract
168. Abstract
169. Abstract
170. Abstract
171. Abstract
172. Abstract
173. Abstract
174. Abstract
175. Abstract
176. Abstract
177. Abstract
178. Abstract
179. Abstract
180. Abstract
181. Abstract
182. Abstract
183. Abstract
184. Abstract
185. Abstract
186. Abstract
187. Abstract
188. Abstract
189. Abstract
190. Abstract
191. Abstract
192. Abstract
193. Abstract
194. Abstract
195. Abstract
196. Abstract
197. Abstract
198. Abstract
199. Abstract
200. Abstract
201. Abstract
202. Abstract
203. Abstract
204. Abstract
205. Abstract
206. Abstract
207. Abstract
208. Abstract
209. Abstract
210. Abstract
211. Abstract
212. Abstract
213. Abstract
214. Abstract
215. Abstract
216. Abstract
217. Abstract
218. Abstract
219. Abstract
220. Abstract
221. Abstract
222. Abstract
223. Abstract
224. Abstract
225. Abstract
226. Abstract
227. Abstract
228. Abstract
229. Abstract
230. Abstract
231. Abstract
232. Abstract
233. Abstract
234. Abstract
235. Abstract
236. Abstract
237. Abstract
238. Abstract
239. Abstract
240. Abstract
241. Abstract
242. Abstract
243. Abstract
244. Abstract
245. Abstract
246. Abstract
247. Abstract
248. Abstract
249. Abstract
250. Abstract
251. Abstract
252. Abstract
253. Abstract
254. Abstract
255. Abstract
256. Abstract
257. Abstract
258. Abstract
259. Abstract
260. Abstract
261. Abstract
262. Abstract
263. Abstract
264. Abstract
265. Abstract
266. Abstract
267. Abstract
268. Abstract
269. Abstract
270. Abstract
271. Abstract
272. Abstract
273. Abstract
274. Abstract
275. Abstract
276. Abstract
277. Abstract
278. Abstract
279. Abstract
280. Abstract
281. Abstract
282. Abstract
283. Abstract
284. Abstract
285. Abstract
286. Abstract
287. Abstract
288. Abstract
289. Abstract
290. Abstract
291. Abstract
292. Abstract
293. Abstract
294. Abstract
295. Abstract
296. Abstract
297. Abstract
298. Abstract
299. Abstract
300. Abstract
301. Abstract
302. Abstract
303. Abstract
304. Abstract
305. Abstract
306. Abstract
307. Abstract
308. Abstract
309. Abstract
310. Abstract
311. Abstract
312. Abstract
313. Abstract
314. Abstract
315. Abstract
316. Abstract
317. Abstract
318. Abstract
319. Abstract
320. Abstract
321. Abstract
322. Abstract
323. Abstract
324. Abstract
325. Abstract
326. Abstract
327. Abstract
328. Abstract
329. Abstract
330. Abstract
331. Abstract
332. Abstract
333. Abstract
334. Abstract
335. Abstract
336. Abstract
337. Abstract
338. Abstract
339. Abstract
340. Abstract
341. Abstract
342. Abstract
343. Abstract
344. Abstract
345. Abstract
346. Abstract
347. Abstract
348. Abstract
349. Abstract
350. Abstract
351. Abstract
352. Abstract
353. Abstract
354. Abstract
355. Abstract
356. Abstract
357. Abstract
358. Abstract
359. Abstract
360. Abstract
361. Abstract
362. Abstract
363. Abstract
364. Abstract
365. Abstract
366. Abstract
367. Abstract
368. Abstract
369. Abstract
370. Abstract
371. Abstract
372. Abstract
373. Abstract
374. Abstract
375. Abstract
376. Abstract
377. Abstract
378. Abstract
379. Abstract
380. Abstract
381. Abstract
382. Abstract
383. Abstract
384. Abstract
385. Abstract
386. Abstract
387. Abstract
388. Abstract
389. Abstract
390. Abstract
391. Abstract
392. Abstract
393. Abstract
394. Abstract
395. Abstract
396. Abstract
397. Abstract
398. Abstract
399. Abstract
400. Abstract
401. Abstract
402. Abstract
403. Abstract
404. Abstract
405. Abstract
406. Abstract
407. Abstract
408. Abstract
409. Abstract
410. Abstract
411. Abstract
412. Abstract
413. Abstract
414. Abstract
415. Abstract
416. Abstract
417. Abstract
418. Abstract
419. Abstract
420. Abstract
421. Abstract
422. Abstract
423. Abstract
424. Abstract
425. Abstract
426. Abstract
427. Abstract
428. Abstract
429. Abstract
430. Abstract
431. Abstract
432. Abstract
433. Abstract
434. Abstract
435. Abstract
436. Abstract
437. Abstract
438. Abstract
439. Abstract
440. Abstract
441. Abstract
442. Abstract
443. Abstract
444. Abstract
445. Abstract
446. Abstract
447. Abstract
448. Abstract
449. Abstract
450. Abstract
451. Abstract
452. Abstract
453. Abstract
454. Abstract
455. Abstract
456. Abstract
457. Abstract
458. Abstract
459. Abstract
460. Abstract
461. Abstract
462. Abstract
463. Abstract
464. Abstract
465. Abstract
466. Abstract
467. Abstract
468. Abstract
469. Abstract
470. Abstract
471. Abstract
472. Abstract
473. Abstract
474. Abstract
475. Abstract
476. Abstract
477. Abstract
478. Abstract
479. Abstract
480. Abstract
481. Abstract
482. Abstract
483. Abstract
484. Abstract
485. Abstract
486. Abstract
487. Abstract
488. Abstract
489. Abstract
490. Abstract
491. Abstract
492. Abstract
493. Abstract
494. Abstract
495. Abstract
496. Abstract
497. Abstract
498. Abstract
499. Abstract
500. Abstract
501. Abstract
502. Abstract
503. Abstract
504. Abstract
505. Abstract
506. Abstract
507. Abstract
508. Abstract
509. Abstract
510. Abstract
511. Abstract
512. Abstract
513. Abstract
514. Abstract
515. Abstract
516. Abstract
517. Abstract
518. Abstract
519. Abstract
520. Abstract
521. Abstract
522. Abstract
523. Abstract
524. Abstract
525. Abstract
526. Abstract
527. Abstract
528. Abstract
529. Abstract
530. Abstract
531. Abstract
532. Abstract
533. Abstract
534. Abstract
535. Abstract
536. Abstract
537. Abstract
538. Abstract
539. Abstract
540. Abstract
541. Abstract
542. Abstract
543. Abstract
544. Abstract
545. Abstract
546. Abstract
547. Abstract
548. Abstract
549. Abstract
550. Abstract
551. Abstract
552. Abstract
553. Abstract
554. Abstract
555. Abstract
556. Abstract
557. Abstract
558. Abstract
559. Abstract
560. Abstract
561. Abstract
562. Abstract
563. Abstract
564. Abstract
565. Abstract
566. Abstract
567. Abstract
568. Abstract
569. Abstract
570. Abstract
571. Abstract
572. Abstract
573. Abstract
574. Abstract
575. Abstract
576. Abstract
577. Abstract
578. Abstract
579. Abstract
580. Abstract
581. Abstract
582. Abstract
583. Abstract
584. Abstract
585. Abstract
586. Abstract
587. Abstract
588. Abstract
589. Abstract
590. Abstract
591. Abstract
592. Abstract
593. Abstract
594. Abstract
595. Abstract
596. Abstract
597. Abstract
598. Abstract
599. Abstract
600. Abstract
601. Abstract
602. Abstract
603. Abstract
604. Abstract
605. Abstract
606. Abstract
607. Abstract
608. Abstract
609. Abstract
610. Abstract
611. Abstract
612. Abstract
613. Abstract
614. Abstract
615. Abstract
616. Abstract
617. Abstract
618. Abstract
619. Abstract
620. Abstract
621. Abstract
622. Abstract
623. Abstract
624. Abstract
625. Abstract
626. Abstract
627. Abstract
628. Abstract
629. Abstract
630. Abstract
631. Abstract
632. Abstract
633. Abstract
634. Abstract
635. Abstract
636. Abstract
637. Abstract
638. Abstract
639. Abstract
640. Abstract
641. Abstract
642. Abstract
643. Abstract
644. Abstract
645. Abstract
646. Abstract
647. Abstract
648. Abstract
649. Abstract
650. Abstract
651. Abstract
652. Abstract
653. Abstract
654. Abstract
655. Abstract
656. Abstract
657. Abstract
658. Abstract
659. Abstract
660. Abstract
661. Abstract
662. Abstract
663. Abstract
664. Abstract
665. Abstract
666. Abstract
667. Abstract
668. Abstract
669. Abstract
670. Abstract
671. Abstract
672. Abstract
673. Abstract
674. Abstract
675. Abstract
676. Abstract
677. Abstract
678. Abstract
679. Abstract
680. Abstract
681. Abstract
682. Abstract
683. Abstract
684. Abstract
685. Abstract
686. Abstract
687. Abstract
688. Abstract
689. Abstract
690. Abstract
691. Abstract
692. Abstract
693. Abstract
694. Abstract
695. Abstract
696. Abstract
697. Abstract
698. Abstract
699. Abstract
700. Abstract
701. Abstract
702. Abstract
703. Abstract
704. Abstract
705. Abstract
706. Abstract
707. Abstract
708. Abstract
709. Abstract
710. Abstract
711. Abstract
712. Abstract
713. Abstract
714. Abstract
715. Abstract
716. Abstract
717. Abstract
718. Abstract
719. Abstract
720. Abstract
721. Abstract
722. Abstract
723. Abstract
724. Abstract
725. Abstract
726. Abstract
727. Abstract
728. Abstract
729. Abstract
730. Abstract
731. Abstract
732. Abstract
733. Abstract
734. Abstract
735. Abstract
736. Abstract
737. Abstract
738. Abstract
739. Abstract
740. Abstract
741. Abstract
742. Abstract
743. Abstract
744. Abstract
745. Abstract
746. Abstract
747. Abstract
748. Abstract
749. Abstract
750. Abstract
751. Abstract
752. Abstract
753. Abstract
754. Abstract
755. Abstract
756. Abstract
757. Abstract
758. Abstract
759. Abstract
760. Abstract
761. Abstract
762. Abstract
763. Abstract
764. Abstract
765. Abstract
766. Abstract
767. Abstract
768. Abstract
769. Abstract
770. Abstract
771. Abstract
772. Abstract
773. Abstract
774. Abstract
775. Abstract
776. Abstract
777. Abstract
778. Abstract
779. Abstract
780. Abstract
781. Abstract
782. Abstract
783. Abstract
784. Abstract
785. Abstract
786. Abstract
787. Abstract
788. Abstract
789. Abstract
790. Abstract
791. Abstract
792. Abstract
793. Abstract
794. Abstract
795. Abstract
796. Abstract
797. Abstract
798. Abstract
799. Abstract
800. Abstract
801. Abstract
802. Abstract
803. Abstract
804. Abstract
805. Abstract
806. Abstract
807. Abstract
808. Abstract
809. Abstract
810. Abstract
811. Abstract
812. Abstract
813. Abstract
814. Abstract
815. Abstract
816. Abstract
817. Abstract
818. Abstract
819. Abstract
820. Abstract
821. Abstract
822. Abstract
823. Abstract
824. Abstract
825. Abstract
826. Abstract
827. Abstract
828. Abstract
829. Abstract
830. Abstract
831. Abstract
832. Abstract
833. Abstract
834. Abstract
835. Abstract
836. Abstract
837. Abstract
838. Abstract
839. Abstract
840. Abstract
841. Abstract
842. Abstract
843. Abstract
844. Abstract
845. Abstract
846. Abstract
847. Abstract
848. Abstract
849. Abstract
850. Abstract
851. Abstract
852. Abstract
853. Abstract
854. Abstract
855. Abstract
856. Abstract
857. Abstract
858. Abstract
859. Abstract
860. Abstract
861. Abstract
862. Abstract
863. Abstract
864. Abstract
865. Abstract
866. Abstract
867. Abstract
868. Abstract
869. Abstract
870. Abstract
871. Abstract
872. Abstract
873. Abstract
874. Abstract
875. Abstract
876. Abstract
877. Abstract
878. Abstract
879. Abstract
880. Abstract
881. Abstract
882. Abstract
883. Abstract
884. Abstract
885. Abstract
886. Abstract
887. Abstract
888. Abstract
889. Abstract
890. Abstract
891. Abstract
892. Abstract
893. Abstract
894. Abstract
895. Abstract
896. Abstract
897. Abstract
898. Abstract
899. Abstract
900. Abstract
901. Abstract
902. Abstract
903. Abstract
904. Abstract
905. Abstract
906. Abstract
907. Abstract
908. Abstract
909. Abstract
910. Abstract
911. Abstract
912. Abstract
913. Abstract
914. Abstract
915. Abstract
916. Abstract
917. Abstract
918. Abstract
919. Abstract
920. Abstract
921. Abstract
922. Abstract
923. Abstract
924. Abstract
925. Abstract
926. Abstract
927. Abstract
928. Abstract
929. Abstract
930. Abstract
931. Abstract
932. Abstract
933. Abstract
934. Abstract
935. Abstract
936. Abstract
937. Abstract
938. Abstract
939. Abstract
940. Abstract
941. Abstract
942. Abstract
943. Abstract
944. Abstract
945. Abstract
946. Abstract
947. Abstract
948. Abstract
949. Abstract
950. Abstract
951. Abstract
952. Abstract
953. Abstract
954. Abstract
955. Abstract
956. Abstract
957. Abstract
958. Abstract
959. Abstract
960. Abstract
961. Abstract
962. Abstract
963. Abstract
964. Abstract
965. Abstract
966. Abstract
967. Abstract
968. Abstract
969. Abstract
970. Abstract
971. Abstract
972. Abstract
973. Abstract
974. Abstract
975. Abstract
976. Abstract
977. Abstract
978. Abstract
979. Abstract
980. Abstract
981. Abstract
982. Abstract
983. Abstract
984. Abstract
985. Abstract
986. Abstract
987. Abstract
988. Abstract
989. Abstract
990. Abstract
991. Abstract
992. Abstract
993. Abstract
994. Abstract
995. Abstract
996. Abstract
997. Abstract
998. Abstract
999. Abstract
1000. Abstract

Abstract

- Abstract
- Abstract
- Abstract

Поддерживаемые сервисы ADS

Ольга Семме

Назад к содержанию

В статье представлен перечень сервисов, входящих в текущую версию ADS. Столбец ***Версия*** содержит ссылки на примечания к релизу каждого компонента.

Сервис	Компонент	Версия
Kafka	Kafka Broker	3.9.1_arenadata1 Kafka Tiered Storage 1.0.0_arenadata1
	Kafka Controller	3.9.1_arenadata1
	Kafka Cruise Control	3.9.1_arenadata1
Kafka Connect	Kafka Connect Worker	3.9.1_arenadata1 Debezium Connector 3.1 Iceberg Sink Connector 1.6.1_arenadata3
Kafka REST Proxy	Kafka REST Proxy	7.9.1
ksqlDB	ksqlDB Client	7.9.1
	ksqlDB Server	7.9.1
Schema-Registry	Schema-Registry	7.9.1
NiFi	NiFi Registry	1.28.1_arenadata2
	NiFi Server	1.28.1_arenadata2
NiFi2	NiFi2 Registry	2.5.0_arenadata1
	NiFi2 Server	2.5.0_arenadata1
MiNiFi	MiNiFi Agent	1.28.1_arenadata2
	MiNiFi C2 Server	1.28.1_arenadata2
ZooKeeper	ZooKeeper Server	3.8.4_arenadata1

Матрица совместимости версий

Ольга Семме

Назад к содержанию

ADS version	ADPS min version	ADPS max version	ADCM min version
3.9.1.1_b1	1.2.1_b1	—	2.7.0
3.9.0.1_b1	1.2.1_b1	—	2.6.0
3.7.2.1_b1	1.2.0_b1	1.2.1_b1	2.5.0
3.6.2.2_b1	1.2.0_b1	1.2.1_b1	2.3.0
3.6.2.1_b1	1.2.0_b1	1.2.1_b1	2.2.0
3.3.2.2_b1	1.1.0_b1	1.2.1_b1	2.0.0
3.3.2.1_b1	1.1.0_b1	1.2.1_b1	2023.10.10.08
2.8.1.1_b1	1.0.3_b1	1.0.4_b6	2023.10.10.08
1.8.2_b1	1.0.3_b1	1.0.4_b6	2023.06.14.16
1.8.1_b1	1.0.3_b1	1.0.4_b6	2022.12.07.10
1.8.0_b1	1.0.3_b1	1.0.4_b6	2022.12.07.10
1.7.2_b2	1.0.3_b1	1.0.4_b6	2022.04.18.13