Yandex Cloud × АЭРО

Самостоятельное развёртывание Lakehouse в Yandex Cloud

Это руководство предназначено для самостоятельного развёртывания и тестирования Lakehouse на базе сервисов Yandex Cloud. Следуя шагам руководства, вы соберёте рабочий аналитический контур, настроите взаимодействие между его компонентами и проверите его работу сквозным приёмочным тестом.

Разворачиваем Lakehouse в Yandex Cloud

В ходе работы вы соберёте аналитический контур Lakehouse из управляемых сервисов Yandex Cloud: объектное хранилище с таблицами Apache Iceberg®, каталог метаданных, два вычислительных движка и оркестратор. В конце вы выполните сквозной приёмочный тест и проверите взаимодействие всех компонентов.

Каждый сервис разворачивается отдельно, но для работы общего контура нужно правильно настроить связи между ними. Например, Apache Spark™ может успешно записать таблицу, но Trino не увидит её из-за разных настроек Metastore. Apache Airflow® может запустить задание, но не подключиться к координатору Trino из-за сетевых ограничений. При этом сами ресурсы будут находиться в рабочем состоянии.

Компоненты Lakehouse используют общее объектное хранилище и каталог метаданных. Поэтому при развёртывании нужно настроить права сервисных аккаунтов, адреса подключений и сетевой доступ между сервисами. После ключевых шагов мы будем проверять не только состояние созданного ресурса, но и его взаимодействие с другими компонентами контура.

Руководство актуально на август 2026 года. Версии сервисов и интерфейс консоли Yandex Cloud со временем меняются, поэтому при развёртывании ориентируйтесь на актуальные версии, доступные в облаке.

Что понадобится

Аккаунт в Yandex Cloud с подключённым платёжным аккаунтом и правами на создание ресурсов в каталоге

Базовое знание SQL и Python® — все команды и листинги приведены в руководстве, но полезно понимать, что выполняет код

SQL-клиент. В руководстве исполь-зуется Yandex WebSQL из консоли Yandex Cloud. Также подойдут DBeaver или DataGrip

Основное время при развёртывании занимает ожидание перехода управляемых сервисов в рабочее состояние.

Важно. Все ресурсы платные и тарифицируются за время работы. Настройте бюджет и уведомления в разделе **Биллинг** до начала работы, останавливайте кластеры в перерывах и удалите ресурсы, когда закончите.

Содержание

Что мы разворачиваем стр. 3–4

Шаг 1. Сервисный аккаунт и роли стр. 5–6

Шаг 2. Бакеты в Yandex Object Storage стр. 7–8

Шаг 3. Сервер Apache Hive™ Metastore стр. 9–10

Шаг 4. Кластер Trino стр. 11–12

Шаг 5. Каталог iceberg и схемы слоёв стр. 13–14

Шаг 6. Кластер Spark стр. 15–16

Шаг 7. Кластер Airflow стр. 17–18

Шаг 8. Сетевой доступ Airflow к Trino стр. 19–21

Шаг 9. Приёмочный тест контура стр. 22–29

Управление расходами стр. 30

Итоги и что делать дальше стр. 30

Что мы разворачиваем

Аналитический контур состоит из пяти основных компонентов. Каждый компонент выполняет свою функцию в архитектуре Lakehouse.

Компонент	Сервис Yandex Cloud	Роль в архитектуре
Объектное хранилище	Yandex Object Storage	Хранит файлы данных и метаданных таблиц Iceberg, код заданий и DAG-файлы
Каталог метаданных	Yandex MetaData Hub, Hive Metastore 3.1	Регистрирует таблицы и хранит ссылки на их актуальные metadata files в Iceberg, позволяя разным движкам работать с одним состоянием таблиц
SQL-движок	Yandex Managed Service for Trino	Выполняет аналитические SQL-запросы, читает и преобразует данные, работает с внешними источниками
Распределённая обработка	Managed Service for Apache Spark™	Выполняет PySpark-задания для очистки, дедупликации и сложных преобразований данных, а также обслуживает таблицы Iceberg
Оркестратор	Managed Service for Apache Airflow®	Запускает задания по расписанию, управляет зависимостями между ними, отслеживает сбои и выполняет повторные попытки

Аналитический контур Lakehouse в Yandex Cloud

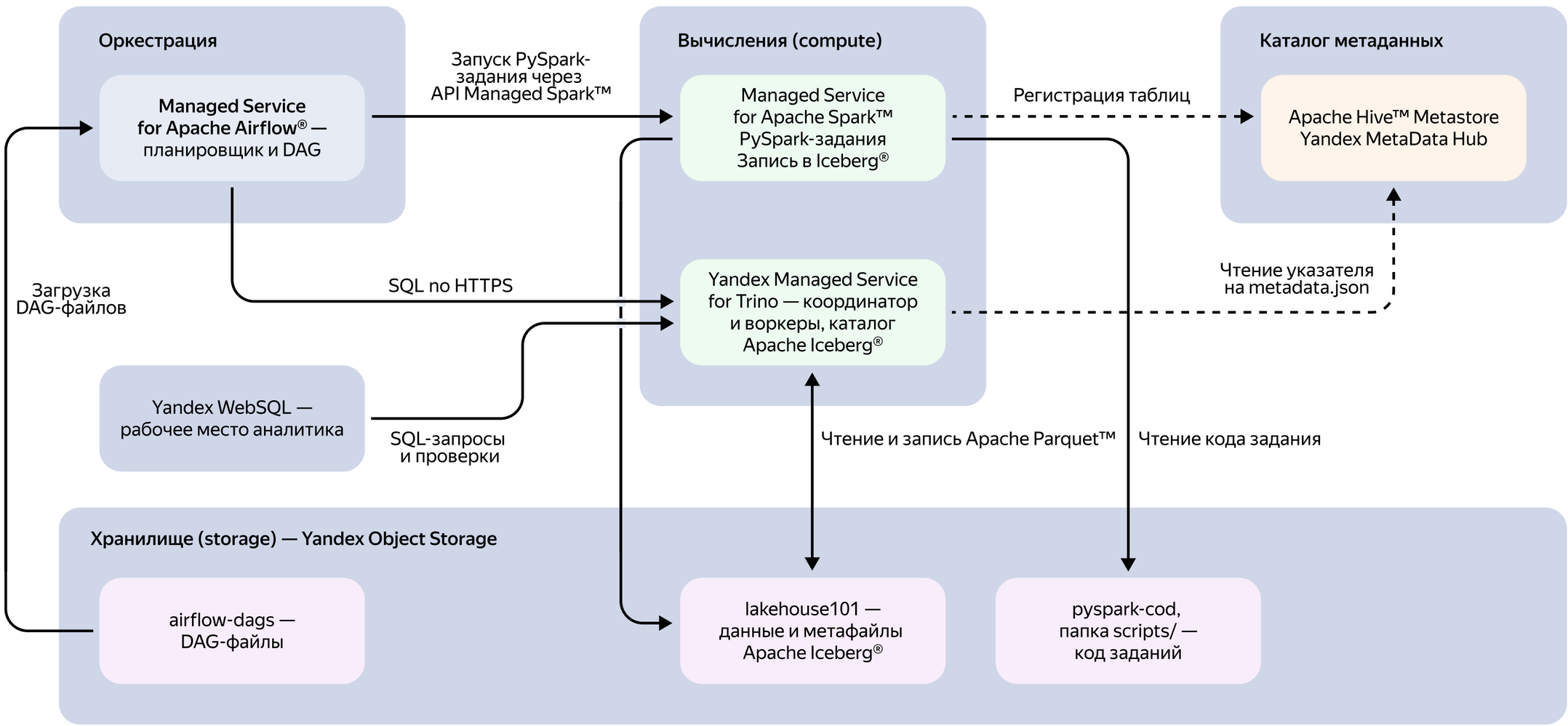


Схема аналитического контура Lakehouse

Порядок развёртывания определяется зависимостями между ресурсами. Metastore использует бакет Yandex Object Storage, каталог `iceberg` в Trino подключается к уже работающему Metastore, а Airflow требует бакет для DAG-файлов, права на запуск Spark-заданий и сетевой доступ к Trino.

Поэтому контур разворачиваем последовательно:

1. Сервисный аккаунт и роли
2. Бакеты в Yandex Object Storage
3. Сервер Hive Metastore
4. Кластер Trino, каталог `iceberg` и схемы слоёв
5. Кластер Spark
6. Кластер Airflow
7. Сетевой доступ Airflow к Trino
8. Приёмочный тест контура

Важно. Все ресурсы Yandex Cloud в этом руководстве тарифицируются. Следите за потреблением в разделе **Биллинг** и останавливайте Trino, Spark, Airflow и Metastore, когда они не используются. Это позволит сохранить созданный контур и снизить расходы между сессиями работы.

Шаг 1 из 9 · 1/2

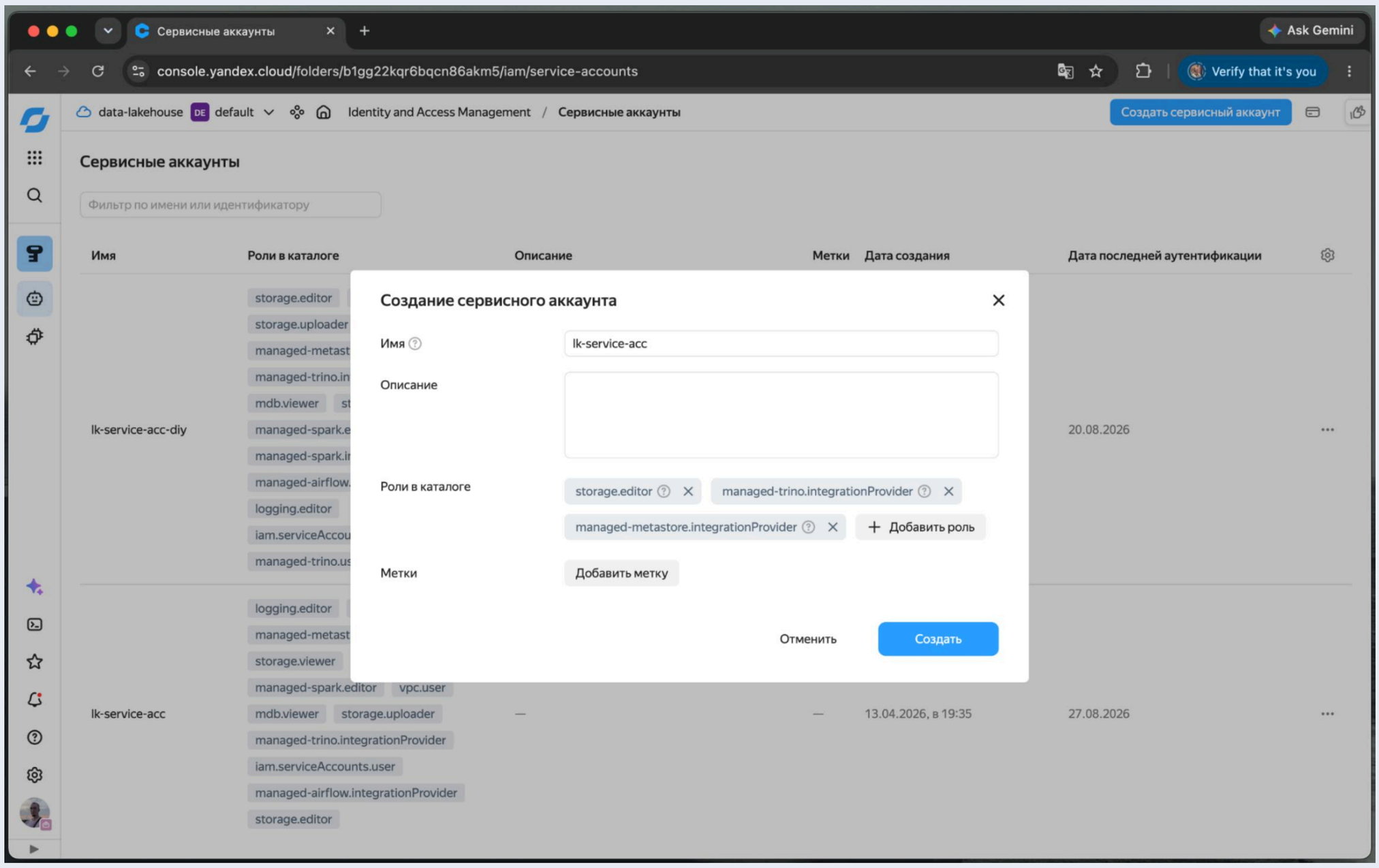
Сервисный аккаунт и роли

Управляемые сервисы Yandex Cloud обращаются к другим ресурсам облака от имени сервисного аккаунта. Например, Trino использует его для доступа к Yandex Object Storage, а Airflow — для запуска заданий в Spark.

Назначенные сервисному аккаунту роли определяют, к каким ресурсам и операциям получат доступ компоненты контура.

В консоли управления перейдите в раздел **Identity and Access Management** и нажмите на **Создать сервисный аккаунт**.

В появившемся окне введите имя аккаунта и назначьте роли.



Назначение ролей сервисному аккаунту

Полный перечень ролей для всего контура приведён ниже. Назначьте их все сразу, чтобы не возвращаться к настройке прав на каждом следующем шаге.

Шаг 1 из 9 · 2/2

Роль	Что она открывает
storage.editor, storage.viewer, storage.uploader	Чтение и запись объектов в бакетах: файлы Iceberg, скрипты PySpark, DAG-файлы
managed-metastore.integrationProvider	Доступ сервера Metastore к ресурсам облака
managed-trino.integrationProvider	Доступ кластера Trino к ресурсам облака
managed-spark.integrationProvider	Доступ кластера Spark к ресурсам облака
managed-airflow.integrationProvider	Доступ кластера Airflow к ресурсам облака
managed-spark.editor	Создание и запуск заданий на кластере Spark из DAG
managed-trino.user	Выполнение запросов в Trino из DAG
iam.serviceAccounts.user	Работа Airflow от имени сервисного аккаунта
vpc.user	Использование подсетей облачной сети
logging.editor, logging.reader	Запись и чтение логов заданий
mdb.viewer	Чтение метаданных управляемых кластеров

Весь набор ролей вешается на один сервисный аккаунт, который используют все компоненты. В продакшен-средах так делать не следует: под каждый сценарий заводят отдельный аккаунт с минимально необходимыми правами, чтобы компрометация одного компонента не давала доступ ко всей платформе. Здесь единый аккаунт сокращает количество шагов настройки.

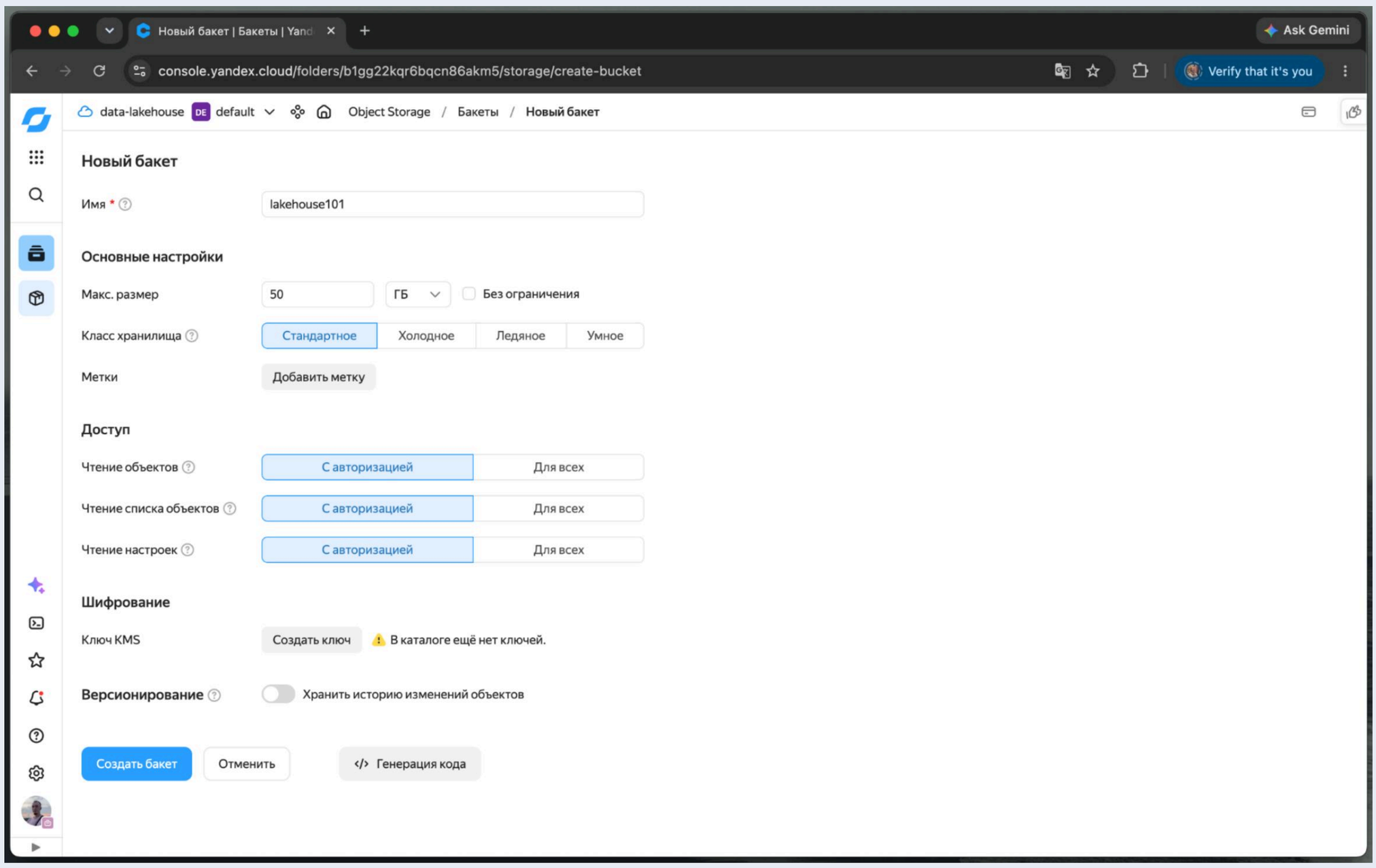
Шаг 2 из 9 · 1/2

Бакеты в Yandex Object Storage

Storage-слой Lakehouse размещается в S3-совместимом объектном хранилище. В Yandex Cloud для этого используется **Yandex Object Storage**. В нём будут храниться Parquet-файлы с данными, метаданные Iceberg, PySpark-скрипты и DAG-файлы Airflow.

Найдите сервис Yandex Object Storage через поиск в консоли управления.

Нажмите на **Создать бакет**, задайте имя и оставьте остальные настройки по умолчанию.



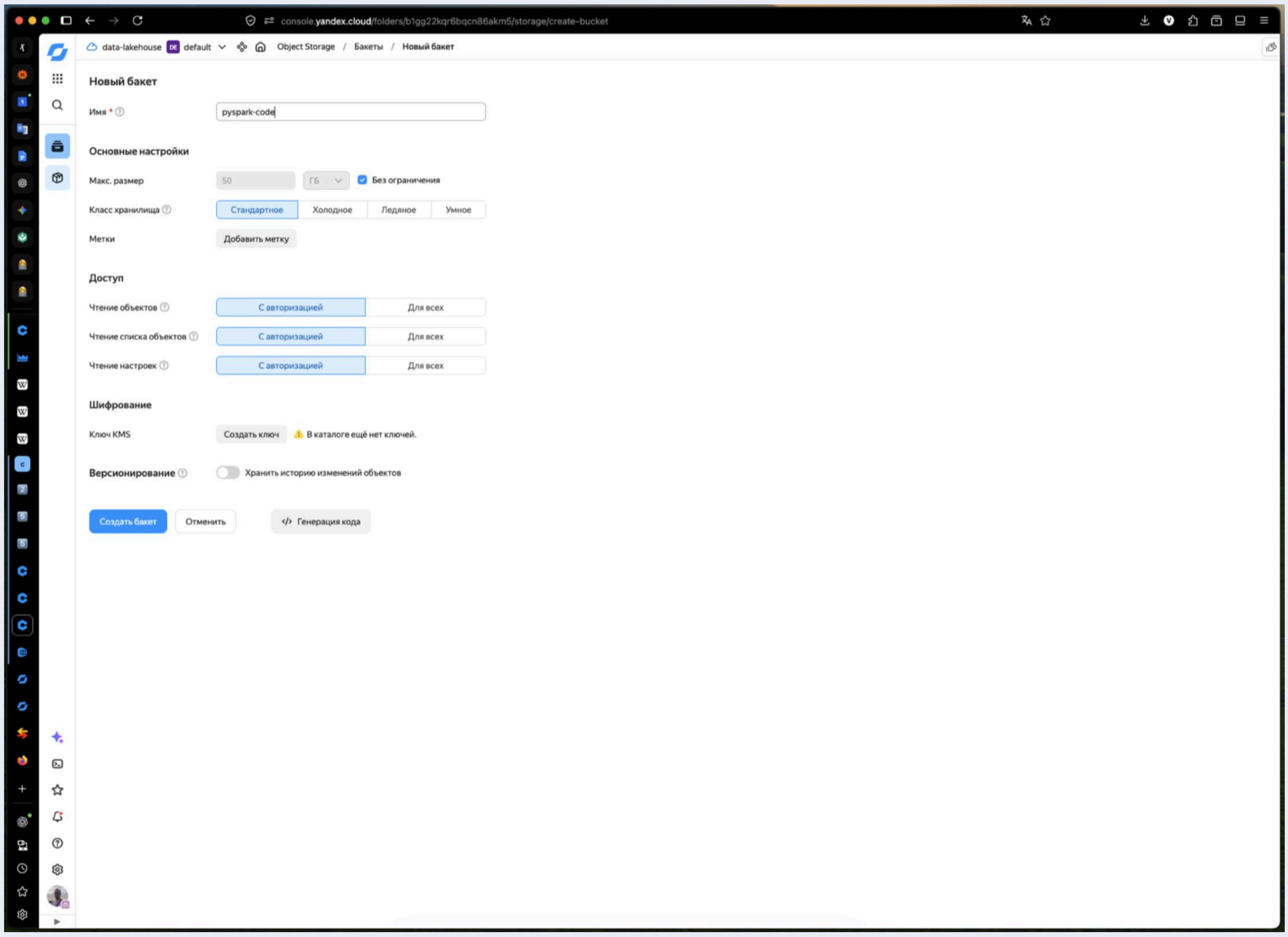
Создание бакета

Создайте три бакета:

- `lakehouse101` — данные и метаданные таблиц Iceberg. В руководстве используется это имя, при выборе другого подставляйте своё во все дальнейшие настройки
- `pyspark-code` — исходный код PySpark-заданий. Внутри создайте папку `scripts`
- `airflow-dags` — DAG-файлы оркестратора

Шаг 2 из 9 · 2/2

Технически всё можно разместить в одном бакете. Разделение по назначению упрощает выдачу прав и жизненный цикл объектов: код и DAG-файлы меняются вместе с релизами, данные живут по своим правилам хранения, а версионирование и политики доступа для них настраиваются по-разному.



Бакет с кодом PySpark

Шаг 3 из 9 · 1/2

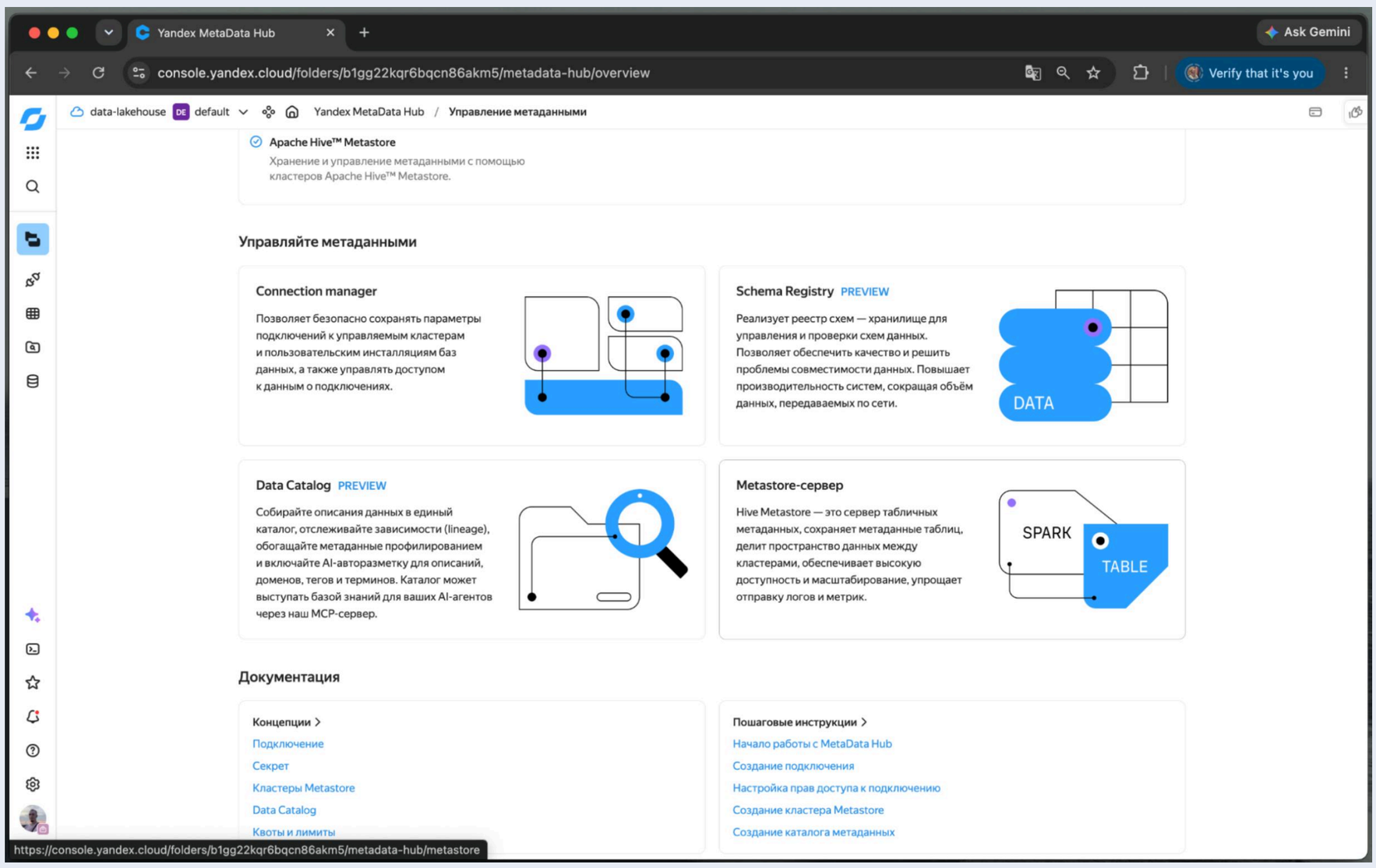
Сервер Apache Hive™ Metastore

Каталог хранит информацию о зарегистрированных таблицах и указывает на актуальные файлы метаданных Iceberg. Благодаря этому разные вычислительные движки могут находить одну и ту же таблицу и работать с её текущим состоянием.

В этом контуре используется управляемый **Hive Metastore** из сервиса Yandex MetaData Hub.

Перейдите в MetaData Hub через поиск в консоли управления.

Выберите раздел **Metastore-сервер** и создайте сервер.

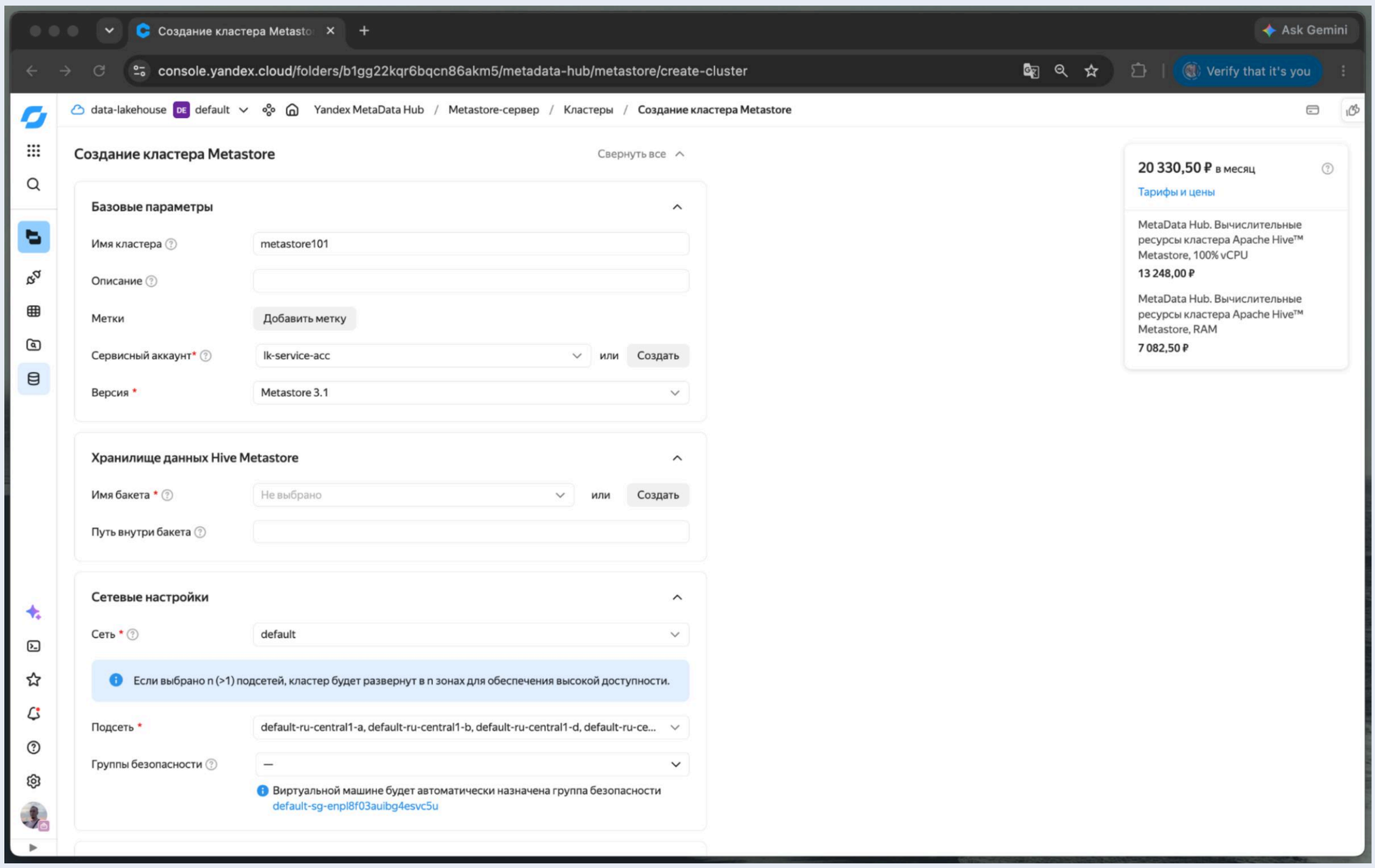


Раздел Metastore-сервер

При создании:

- задайте имя сервера
- подключите созданный сервисный аккаунт
- выберите версию **Metastore 3.1**
- в разделе **Хранилище данных Hive Metastore** укажите бакет `lakehouse101`
- в разделе **Metastore** оставьте тип `cpu-optimized` с пресетом `c2-m4` — 2 vCPU и 4 ГБ памяти
- в разделе **Логирование** включите запись логов

Шаг 3 из 9 · 2/2



Хранилище данных Hive Metastore

Дождитесь, когда сервер перейдёт в статус **Alive**, и запишите его IP-адрес — он указан в строке созданного сервера в разделе Hive Metastore. Адрес понадобится на следующем шаге.

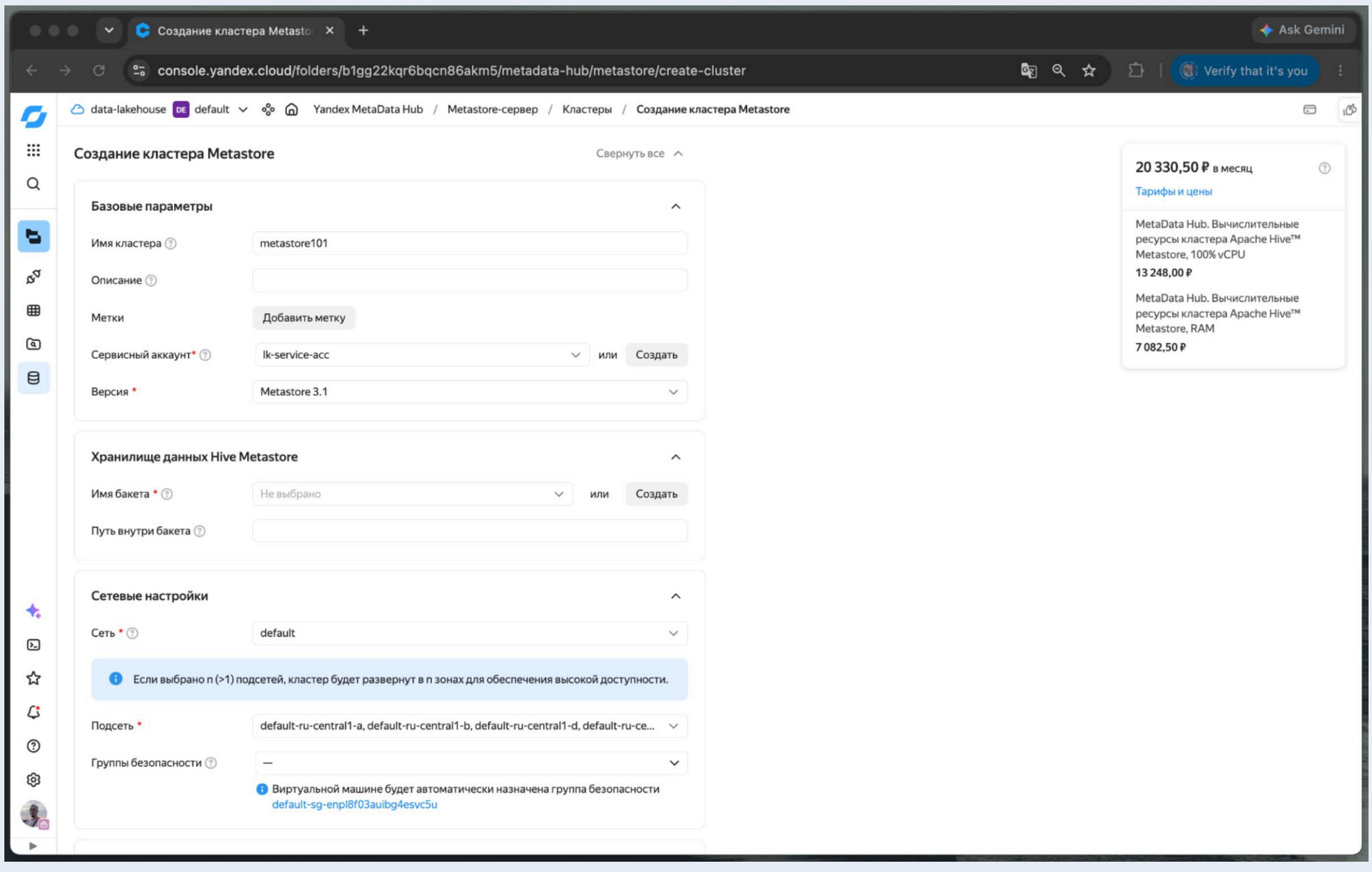
Шаг 4 из 9 · 1/2

Кластер Trino

Trino в контуре решает две задачи: выполняет аналитические запросы к Iceberg и работает федеративным движком, подключая внешние источники как каталоги.

Нажмите на **Создать ресурс** в правом верхнем углу консоли и выберите **Кластер Trino**.

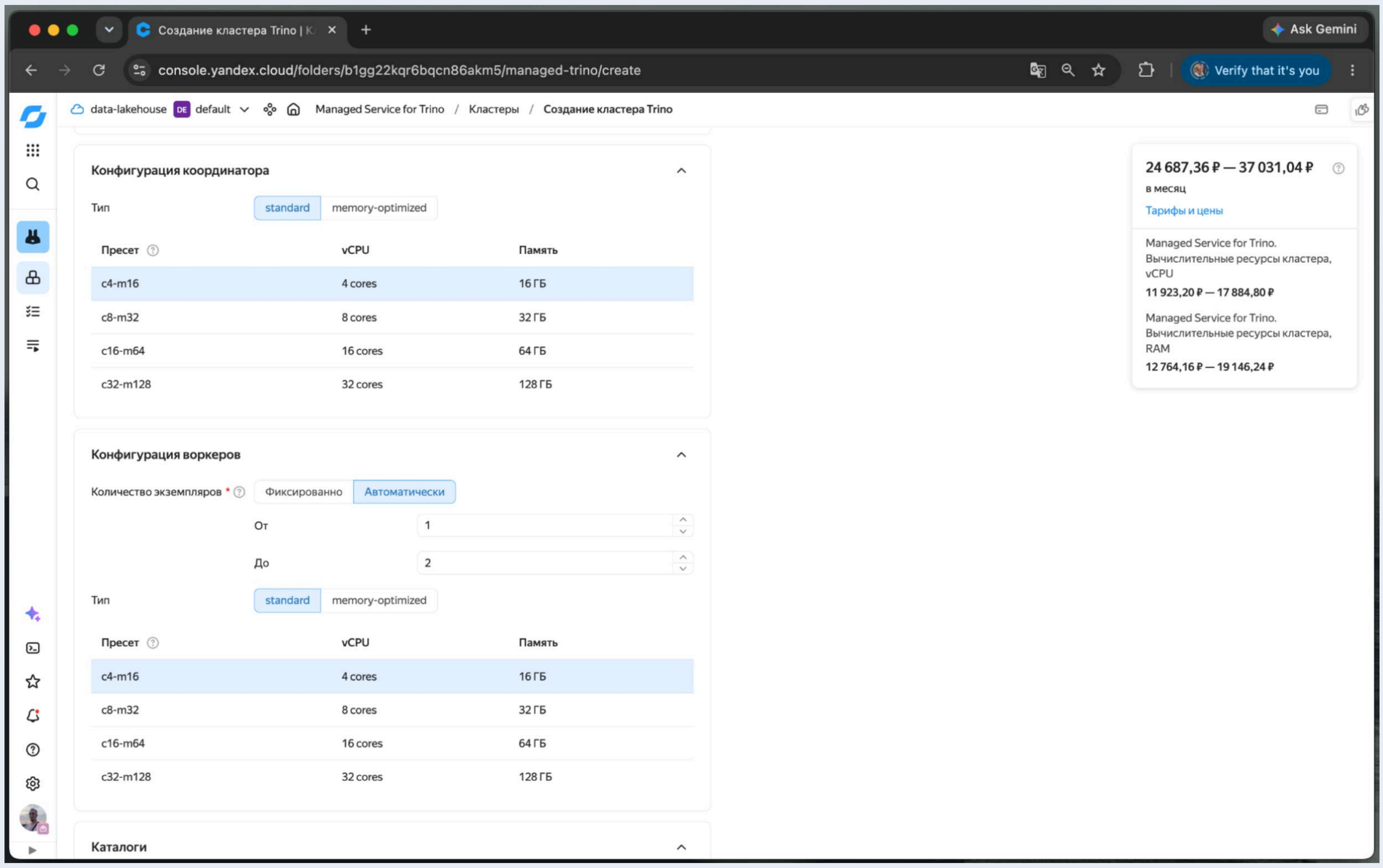
В разделе **Базовые параметры** задайте имя кластера, выберите версию и подключите сервисный аккаунт.



Базовые параметры кластера Trino

Блоки **Сетевые настройки** и **Политика перезапросов** оставьте без изменений. В разделе **Конфигурация координатора** выберите тип standard и пресет c4-m16 (4 vCPU, 16 ГБ). В разделе **Конфигурация воркеров** задайте количество экземпляров **Автоматически** в диапазоне от 1 до 2, тип standard, пресет c4-m16.

Шаг 4 из 9 · 2/2



Конфигурация координатора и воркеров

Дождитесь статуса **Alive**. После этого проверьте доступность кластера: откройте Yandex WebSQL — сервис доступен из консоли. Кластер Trino появится в списке подключений автоматически — или подключитесь другим SQL-клиентом.

Выполните запрос:

```
SELECT version();
```

Ответом будет номер версии кластера, например 476. Кластер работает и принимает запросы.

Шаг 5 из 9 · 1/2

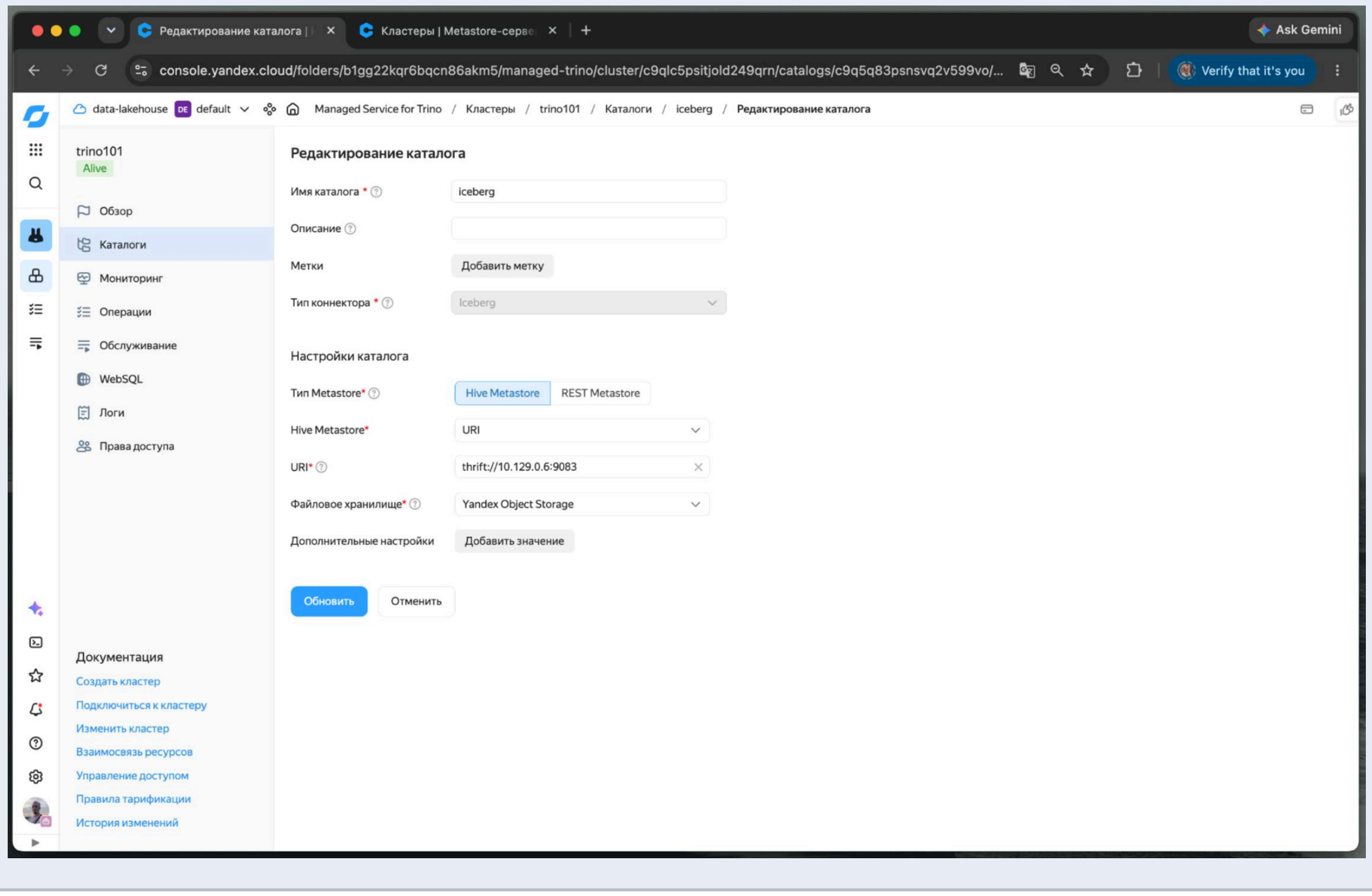
Каталог iceberg и схемы слоёв

Теперь подключим Trino к таблицам Iceberg в объектном хранилище.

В Trino **каталог** — это настроенное подключение к источнику данных. Для работы с Iceberg создадим каталог `iceberg`: через него Trino обращается к Hive Metastore, получает информацию о зарегистрированных таблицах и их актуальных метаданных, а затем читает соответствующие файлы из Yandex Object Storage.

Откройте созданный кластер Trino и перейдите во вкладку **Каталоги**.

Создайте каталог с названием iceberg и типом коннектора Iceberg. Тип Metastore — Hive Metastore, файловое хранилище — Yandex Object Storage. URI подключения указывается в формате thrift://<IP-адрес>:<порт>, где IP-адрес взят из карточки Metastore-сервера, а порт равен 9083.



Создание каталога iceberg

Дождитесь применения настроек. В левой панели Yandex WebSQL в списке каталогов появится `iceberg`.

Шаг 5 из 9 · 1/2

Создание тестовой схемы

Создадим схему `sandbox`, которую будем использовать для проверки работы контура.

Выполните в Yandex WebSQL:

```
CREATE SCHEMA iceberg.sandbox;
```

Обновите список схем в левой панели Yandex WebSQL — схема `sandbox` появится внутри каталога `iceberg`.

Затем откройте бакет `lakehouse101`. После создания схемы в нём появится каталог вида `internal/sandbox.db`. Этот путь связан со схемой `sandbox`: в дальнейшем внутри него будут размещаться файлы таблиц, которые вы создадите в этой схеме.

На следующих шагах будем использовать `sandbox` для тестовых таблиц и проверки взаимодействия компонентов Lakehouse.

Шаг 6 из 9 · 1/2

Кластер Spark

Spark используется для задач распределённой обработки данных, которые удобнее реализовывать на PySpark: сложной очистки, дедупликации, преобразования полуструктурированных данных и обслуживания таблиц Iceberg.

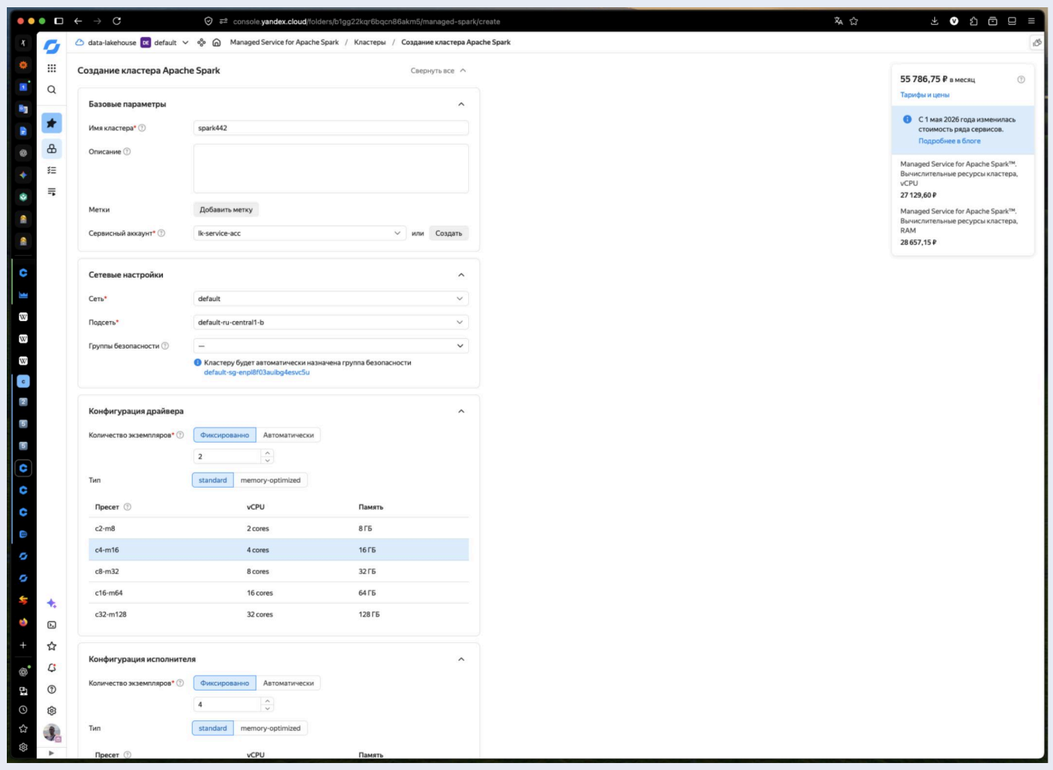
Перейдите в сервис **Managed Service for Apache Spark™** и создайте кластер Apache Spark.

В разделе **Базовые параметры** задайте имя кластера и подключите созданный сервисный аккаунт.

Базовые параметры кластера Spark

В разделе **Конфигурация драйвера** задайте фиксированное количество экземпляров — 2, тип — standard, пресет — c4-m16 (4 vCPU, 16 ГБ). В разделе **Конфигурация исполнителя** выберите **Автоматически** в диапазоне от 0 до 4, тип standard, тот же пресет.

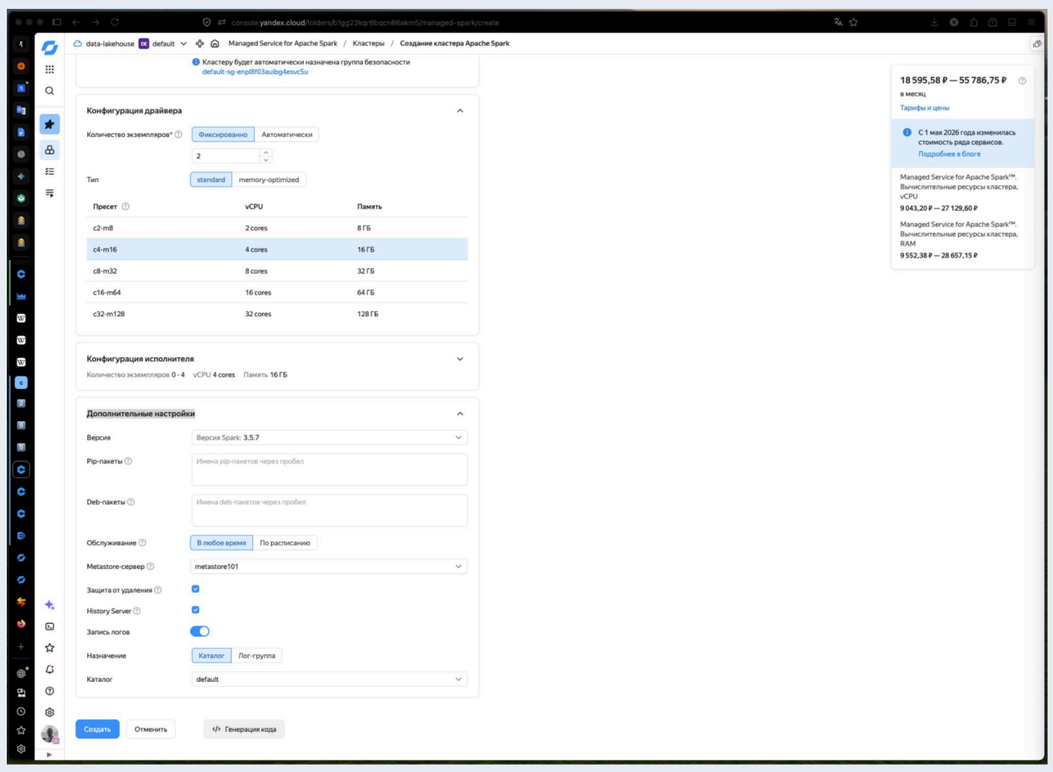
Шаг 6 из 9 · 2/2



Конфигурация драйвера и исполнителя

В разделе **Дополнительные настройки** задайте три параметра:

- Версия Spark — 3.5.7 или наиболее свежая из доступных
- **Укажите Metastore-сервер, созданный на шаге 3. Spark будет регистрировать созданные таблицы в том же каталоге, который использует Trino**
- Включите History Server и запись логов — они понадобятся при разборе упавших заданий



Дополнительные настройки кластера Spark

Дождитесь, пока кластер перейдёт в статус `Running`. Создание занимает несколько минут.

Отдельным ручным заданием Spark мы проверять не будем: его работоспособность вместе с остальными связями проверит приёмочный тест на шаге 9.

Шаг 7 из 9 · 1/2

Кластер Airflow

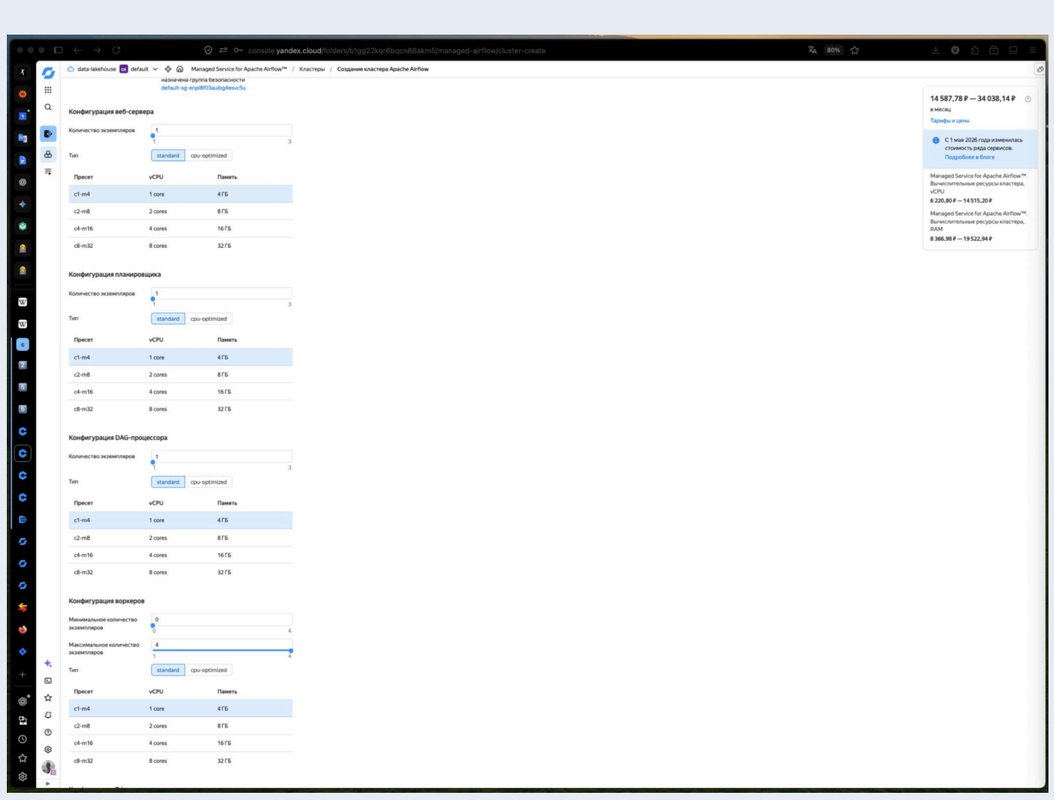
Airflow будет оркестрировать обработку данных: запускать задания, управлять зависимостями между ними, отслеживать сбои и выполнять повторные попытки.

Перейдите в сервис **Managed Service for Apache Airflow®** и нажмите на **Создать**.

В основном разделе выполните три действия:

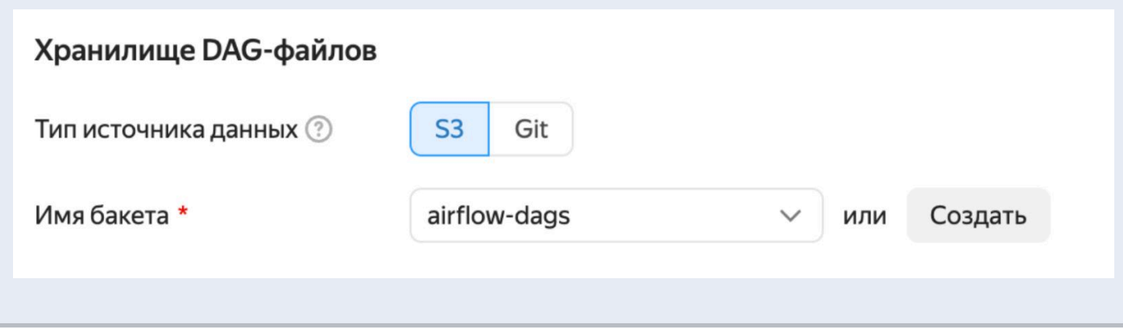
- Выберите версию кластера Airflow 3.1 (или наиболее свежую из доступных) и версию Python 3.12
- Задайте пароль пользователя admin — он понадобится для входа в веб-интерфейс — и сохраните его
- Подключите сервисный аккаунт

В конфигурациях веб-сервера, планировщика и DAG-процессора оставьте по одному экземпляру типа `standard` с пресетом `c1-m4` (1 vCPU, 4 ГБ). Для воркеров задайте диапазон от 0 до 4 экземпляров того же пресета.



Конфигурация кластера Airflow

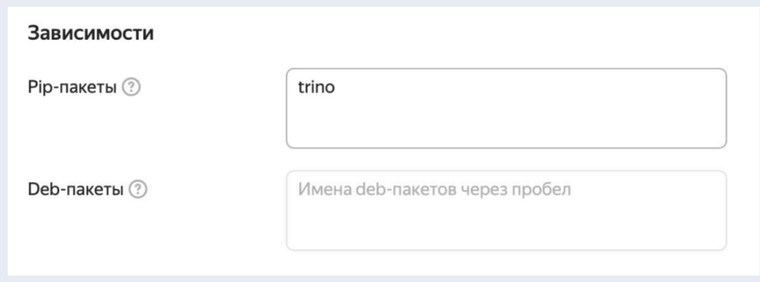
В разделе **Хранилище DAG-файлов** выберите бакет `airflow-dags`.



Хранилище DAG-файлов

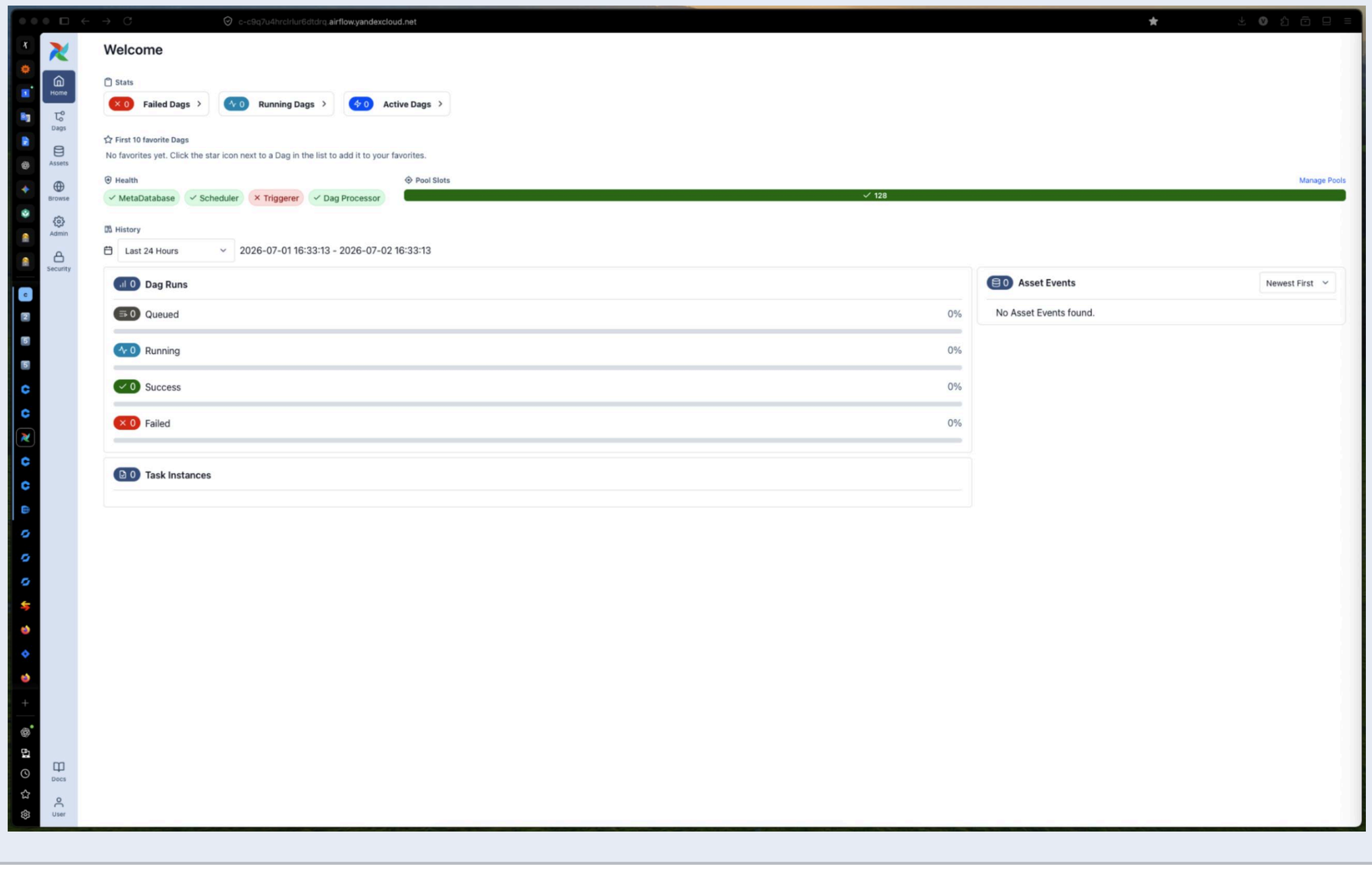
Шаг 7 из 9 · 2/2

В разделе **Зависимости** в поле **Pip-пакеты** добавьте пакет `trino` . Он понадобится DAG'у приёмочного теста, чтобы открыть соединение с координатором Trino. Без этого пакета DAG не пройдёт парсинг и не появится в интерфейсе.



Добавление pip-пакета Trino

Создайте кластер и дождитесь статуса `Alive.` Затем откройте карточку кластера и перейдите по ссылке веб-сервера — откроется главная страница Airflow.



Веб-интерфейс Airflow

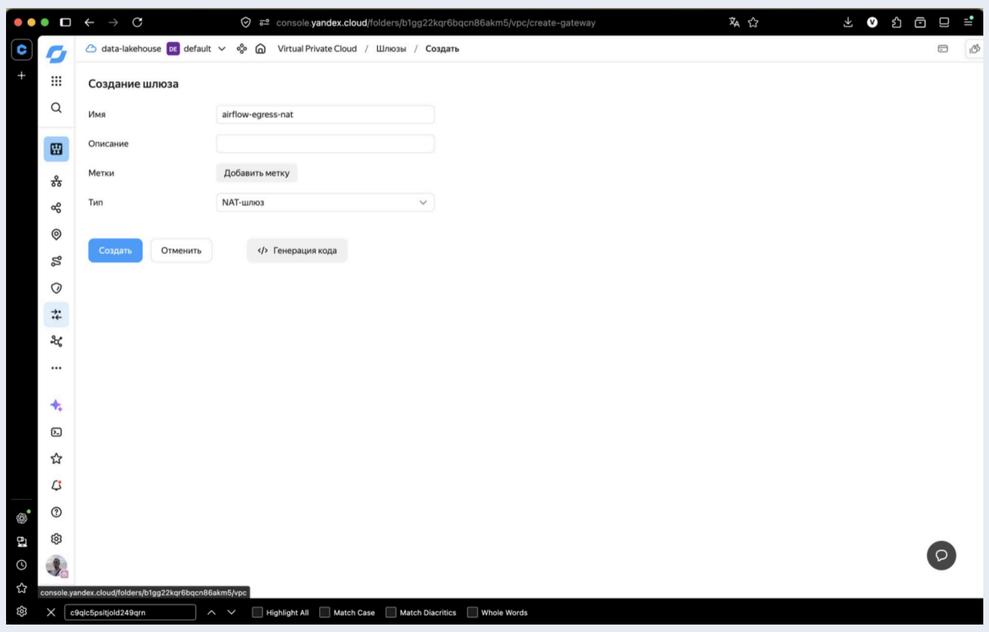
Шаг 8 из 9 · 1/3

Сетевой доступ Airflow к Trino

Airflow запускает задания через внутренний API Yandex Cloud, а к координатору Trino обращается по HTTPS как обычный клиент. Подсети Managed Service for Airflow® по умолчанию не имеют исходящего маршрута в интернет, поэтому такое соединение не установится — задача повиснет и упадёт по тайм-ауту. Настроим маршрут.

Перейдите в раздел **Шлюзы** сервиса **Yandex Virtual Private Cloud** и создайте шлюз:

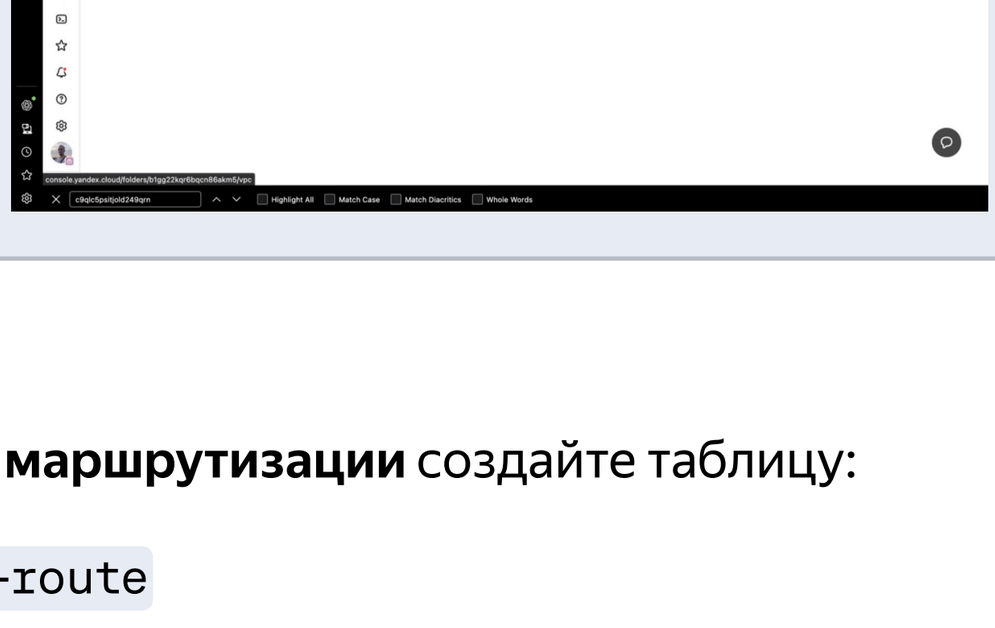
- Имя — `airflow-egress-nat`
- Тип — NAT-шлюз



Создание NAT-шлюза

Затем в разделе **Таблицы маршрутизации** создайте таблицу:

- Имя — `airflow-egress-route`
- Сеть — `default`

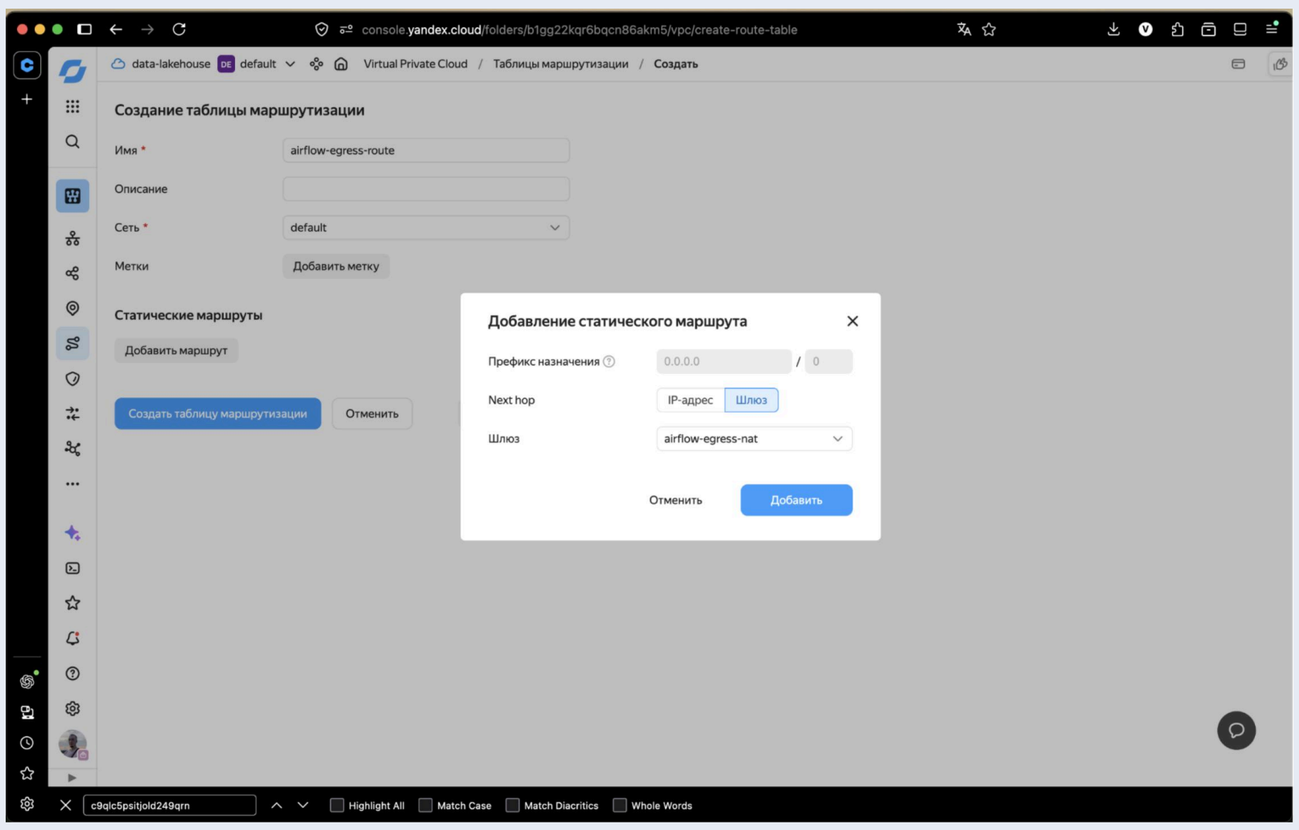


Создание таблицы маршрутизации

Шаг 8 из 9 · 2/3

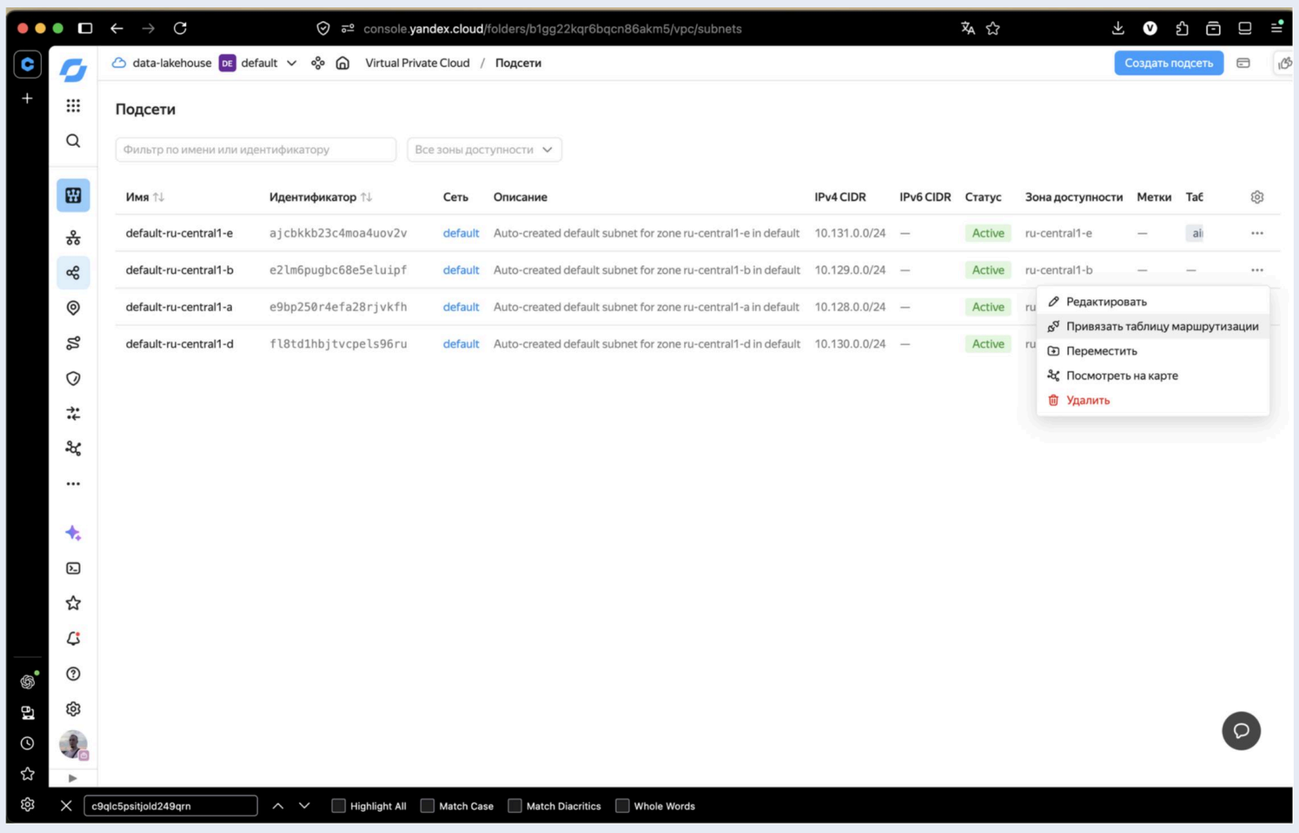
Добавьте в таблицу маршрут:

- Префикс назначения — `0.0.0.0/0`
- Next hop — **Шлюз**
- Шлюз — `airflow-egress-nat`



Маршрут через NAT-шлюз

Привяжите таблицу маршрутизации ко всем подсетям, которые использует кластер Airflow. Перейдите в раздел **Подсети** и укажите таблицу для каждой из них.

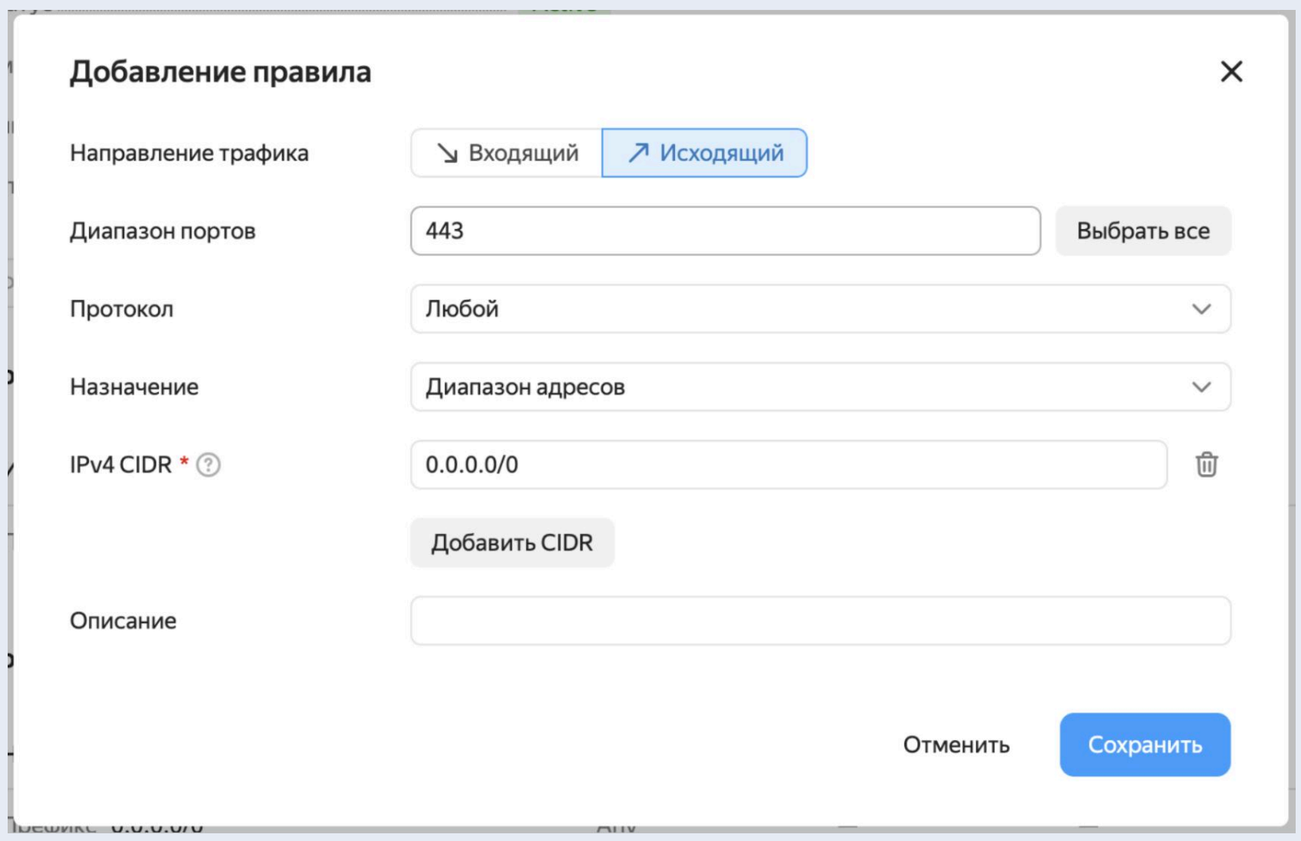


Привязка таблицы маршрутизации к подсетям

Шаг 8 из 9 · 3/3

Последнее — правило исходящего трафика. Откройте раздел **Группы безопасности**, выберите группу кластера Airflow и добавьте правило:

- Направление трафика — исходящий
- Диапазон портов — `443`
- Протокол — любой
- Назначение — диапазон адресов
- IPv4 CIDR — `0.0.0.0/0`



Правило исходящего трафика на порт 443

Маршрут и правило группы безопасности работают вместе: таблица маршрутизации определяет, куда пакет пойдёт, а группа безопасности разрешает ему выйти. Отсутствие любого из двух элементов приводит к одинаковому симптому — тайм-ауту при подключении к Trino.

Шаг 9 из 9 · 1/8

Приёмочный тест контура

Все компоненты развёрнуты. Теперь проверим их совместную работу сквозным тестом.

Тест проходит через основные компоненты контура:

- Airflow запускает PySpark-задание в Managed Service for Apache Spark™
- Spark читает скрипт из бакета `pyspark-code`, создаёт таблицу Iceberg `sandbox.events`, записывает данные в `lakehouse101` и регистрирует таблицу в Hive Metastore
- Airflow подключается к Trino по HTTPS
- Trino читает ту же таблицу через каталог `iceberg`
- После проверки Trino удаляет тестовую таблицу

Все компоненты развёрнуты. Теперь проверим их совместную работу сквозным тестом.

Скрипт записи

Создайте файл `smoke_write.py` со следующим кодом и загрузите его в бакет `pyspark-code` в папку `scripts/`.

```
"""
Приёмочный тест контура Lakehouse: запись данных через Spark.
"""

import random

from pyspark.sql import SparkSession


# Схема sandbox была создана ранее при настройке каталога iceberg.
DB = "sandbox"
TABLE = "events"

# Размер каждой из двух порций данных.
BATCH_SIZE = 10


def main() -> None:
   spark = (
       SparkSession.builder
       .appName("lakehouse_smoke_write")
       .enableHiveSupport()
       .getOrCreate()
   )

   # Таблица пересоздаётся при каждом запуске,
   # поэтому тест можно выполнять повторно.
   spark.sql(f"DROP TABLE IF EXISTS {DB}.{TABLE}")
   spark.sql(
       f"""
       CREATE TABLE {DB}.{TABLE} (
           id BIGINT,
           value DOUBLE,
           producer STRING
       ) USING iceberg
       """
   )

   first_batch = spark.createDataFrame(
       [(i, random.random(), "spark") for i in range(BATCH_SIZE)],
       ["id", "value", "producer"],
   )
   first_batch.writeTo(f"{DB}.{TABLE}").append()
```

```
# Сохраняем snapshot после первой записи.
snapshot_after_first = spark.sql(
    f"SELECT max(snapshot_id) FROM {DB}.{TABLE}.snapshots"
).collect()[0][0]

second_batch = spark.createDataFrame(
    [
        (i, random.random(), "spark")
        for i in range(BATCH_SIZE, BATCH_SIZE * 2)
    ],
    ["id", "value", "producer"],
)
second_batch.writeTo(f"{DB}.{TABLE}").append()

rows_now = spark.table(f"{DB}.{TABLE}").count()
rows_before = spark.sql(
    f"SELECT COUNT(*) FROM {DB}.{TABLE} "
    f"VERSION AS OF {snapshot_after_first}"
).collect()[0][0]

print(
    f"SMOKE WRITE OK | table={DB}.{TABLE} | "
    f"rows_now={rows_now} | rows_before={rows_before}",
    flush=True,
)

spark.stop()


if __name__ == "__main__":
    main()
```

Скрипт записывает данные двумя порциями по 10 строк. После первой записи он сохраняет идентификатор snapshot, а затем добавляет ещё 10 строк.

В результате Spark должен увидеть:

- **20 строк** в текущем состоянии таблицы
- **10 строк** при чтении snapshot после первой записи

Так мы одновременно проверяем запись в Iceberg и возможность обращаться к предыдущему состоянию таблицы.

DAG приёмочного теста

Создайте файл `lakehouse_smoke_test_dag.py`.

Перед загрузкой укажите в константах:

- идентификатор кластера Spark
- имя бакета с PySpark-кодом
- имя бакета Lakehouse
- FQDN координатора Trino из карточки кластера

Шаг 9 из 9 · 3/8

```
"""
Приёмочный тест контура Lakehouse.
"""

import logging
from contextlib import closing
from datetime import timedelta

import pendulum
import yandexcloud
from airflow.providers.yandex.hooks.yandex import YandexCloudBaseHook
from airflow.sdk import dag, task
from trino.auth import BasicAuthentication
from trino.dbapi import connect
from yandexcloud.operations import OperationError


# Идентификатор кластера Managed Service for Apache Spark.
SPARK_CLUSTER_ID = "<идентификатор_кластера_Spark>"

# Путь к PySpark-скрипту.
SPARK_SCRIPT_URI = (
   "s3a://<имя_бакета_с_кодом>/scripts/smoke_write.py"
)

# Warehouse, в котором Spark размещает файлы таблиц Iceberg.
SPARK_PROPERTIES = {
   "spark.sql.warehouse.dir":
       "s3a://<имя_бакета_Lakehouse>/internal/",
}

# FQDN координатора (идентификатор) Managed Service for Trino. (не забудьте префикс "c-")
TRINO_HOST = "c-<хост_координатора>.trino.yandexcloud.net"

# Тестовая таблица.
TEST_SCHEMA = "sandbox"
TEST_TABLE = "events"

EXPECTED_ROWS = 20


def run_trino_query(sql: str) -> list:
   """Выполняет SQL-запрос в Managed Trino."""
   sdk = yandexcloud.SDK()
   iam_token = sdk._channels._token_requester.get_token()

   with closing(
       connect(
           host=TRINO_HOST,
           port=443,
           user="iam",
           auth=BasicAuthentication("iam", iam_token),
           http_scheme="https",
           request_timeout=30,
       )
   ) as connection:
       with closing(connection.cursor()) as cursor:
           cursor.execute(sql)
           return cursor.fetchall()


@dag(
   dag_id="lakehouse_smoke_test",
   description=(
       "Airflow запускает Spark, "
       "после чего Trino читает записанные данные"
   ),
   schedule=None,
   start_date=pendulum.datetime(
       2026, 1, 1, tz="Europe/Moscow"
   ),
   catchup=False,
   max_active_runs=1,
   default_args={
       "retries": 1,
       "retry_delay": timedelta(minutes=5),
   },
   tags=["lakehouse", "smoke-test"],
)
def lakehouse_smoke_test():

   @task
   def spark_write() -> str:
       """Запускает PySpark-задание и ожидает его завершения."""
       yc_hook = YandexCloudBaseHook()
       spark_client = yc_hook.sdk.wrappers.Spark()
```

```
    job_spec = yc_hook.sdk.wrappers.PysparkJobParameters(
        name="lakehouse_smoke_write",
        main_python_file_uri=SPARK_SCRIPT_URI,
        properties=SPARK_PROPERTIES,
    )

    job_id = None

    try:
        logging.info("Запускаем PySpark-задание smoke_write")

        operation = spark_client.create_pyspark_job(
            cluster_id=SPARK_CLUSTER_ID,
            spec=job_spec,
        )
        job_id = operation.response.id

        logging.info(
            "Spark-job %s завершена успешно",
            job_id,
        )
        return job_id

    except OperationError as error:
        job_id = error.operation_result.meta.job_id
        logging.exception(
            "Spark-job завершилась ошибкой"
        )
        raise

    finally:
        if job_id:
            logging.info(
                "Логи Spark-job %s:",
                job_id,
            )
            for line in spark_client.get_job_log(
                cluster_id=SPARK_CLUSTER_ID,
                job_id=job_id,
            ):
                logging.info(line)

@task
def trino_check(spark_job_id: str) -> int:
    """Читает записанную Spark таблицу через Trino."""
    logging.info(
        "Проверяем результат Spark-job %s средствами Trino",
        spark_job_id,
    )

    row_count = run_trino_query(
        f"SELECT COUNT(*) "
        f"FROM iceberg.{TEST_SCHEMA}.{TEST_TABLE}"
    )[0][0]

    if row_count != EXPECTED_ROWS:
        raise ValueError(
            f"Trino видит {row_count} строк "
            f"вместо {EXPECTED_ROWS}. "
            "Проверьте настройки Metastore "
            "и Object Storage."
        )

    snapshot_count = run_trino_query(
        f'SELECT COUNT(*) '
        f'FROM iceberg.{TEST_SCHEMA}.'
        f'"{TEST_TABLE}$snapshots"'
    )[0][0]

    logging.info(
        "Trino читает таблицу iceberg.%s.%s: "
        "строк %s, снимков %s",
        TEST_SCHEMA,
        TEST_TABLE,
        row_count,
        snapshot_count,
    )

    return row_count

trino_check(spark_write())


lakehouse_smoke_test()
```

DAG состоит из двух последовательных задач:

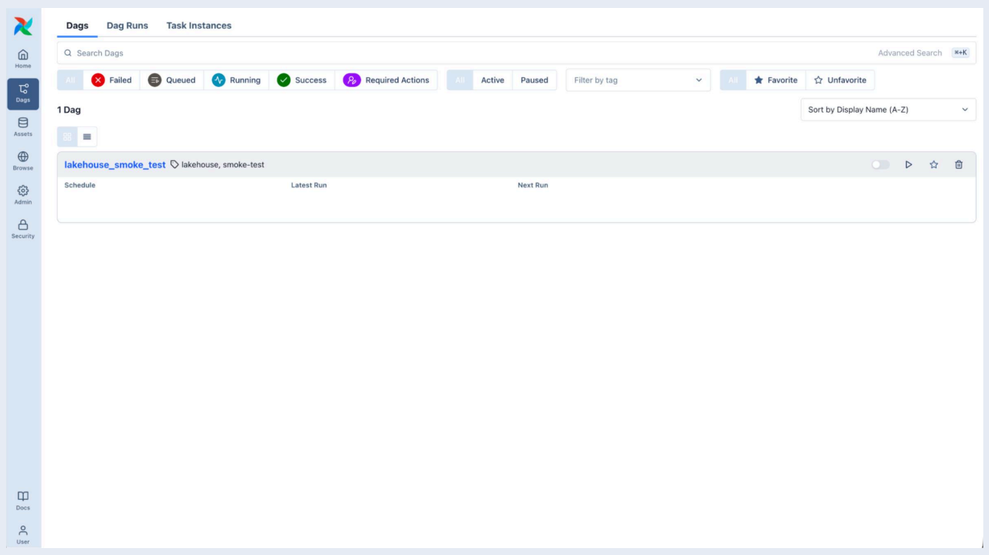
1. `spark_write` запускает PySpark-задание и передаёт его идентификатор следующей задаче
2. `trino_check` читает созданную Spark-таблицу через Trino и проверяет количество строк и snapshots

После завершения теста таблица `sandbox.events` остаётся в каталоге `iceberg`, поэтому её можно открыть и исследовать через Yandex WebSQL

Параметр `schedule=None` означает, что DAG не запускается по расписанию. Для приёмочного теста достаточно выполнить его вручную после развёртывания контура

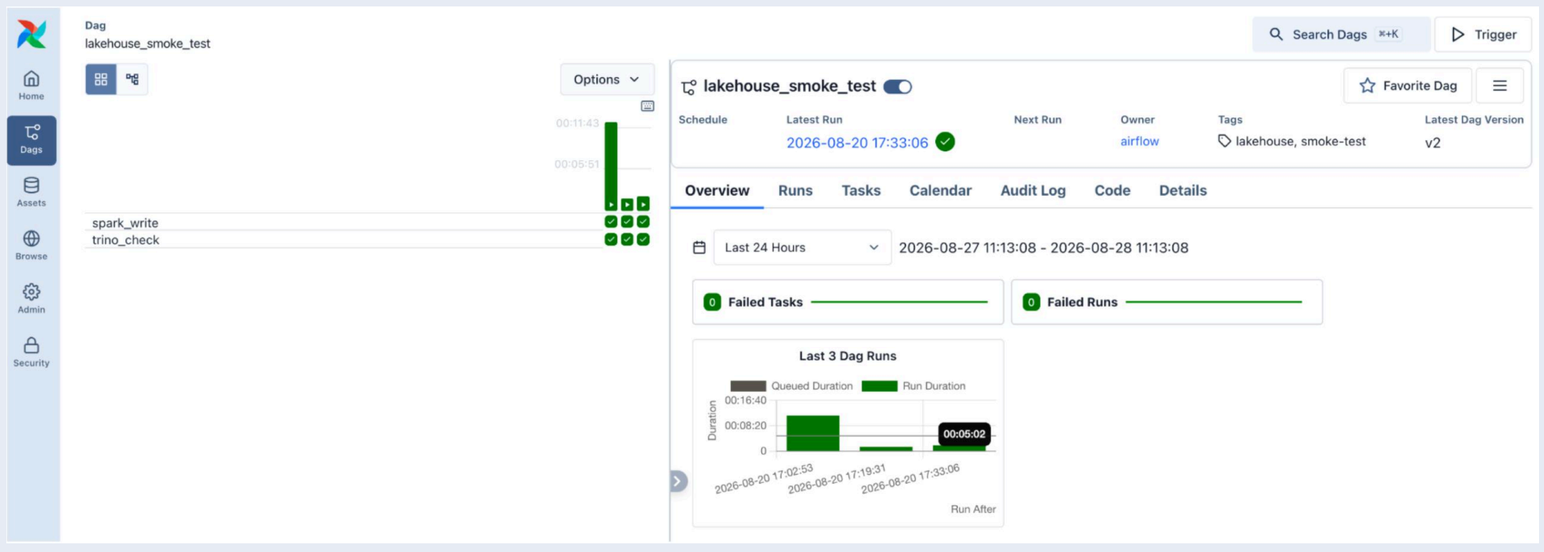
Запуск

Загрузите файл `lakehouse_smoke_test_dag.py` в бакет `airflow-dags`. Airflow подхватывает новые файлы не мгновенно, поэтому подождите, пока DAG появится в разделе DAGs.



Раздел DAGs в интерфейсе Airflow

Новый DAG загружается в неактивном состоянии. Откройте его и нажмите **Trigger** для ручного запуска — после первого запуска DAG активируется автоматически.

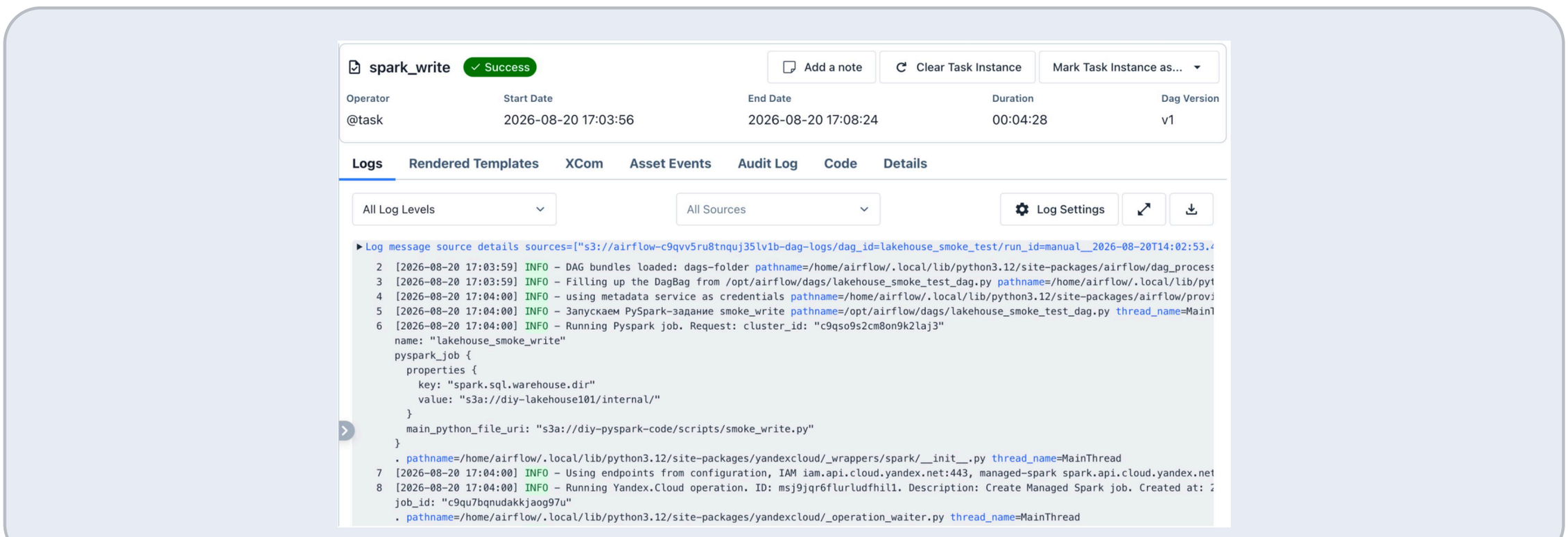


Кнопка Trigger для ручного запуска

Шаг 9 из 9 · 6/8

Выполнение приёмочного теста может занять несколько минут: Airflow запускает отдельное задание Spark, ожидает его завершения, а затем выполняет проверку через Trino. Продолжительность зависит от времени запуска вычислительных ресурсов и текущей нагрузки на сервисы.

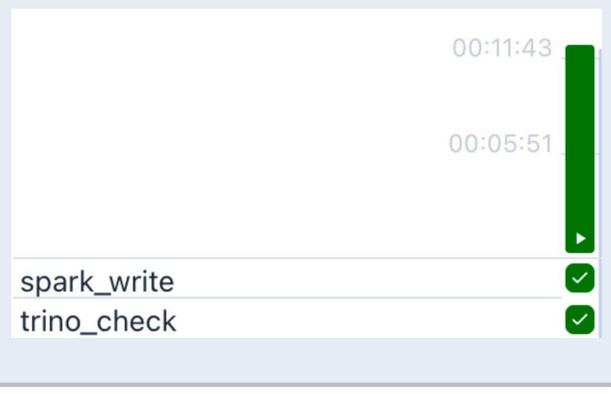
Следите за ходом выполнения на странице DAG. Чтобы открыть журнал конкретной задачи, выберите её на графе и перейдите во вкладку **Logs**.



Вкладка Logs с журналом выполнения задачи

При успешном прохождении теста:

- задача `spark_write` завершится в состоянии **Success**, а в её логе будет видно успешное завершение Spark-job
- в логе задачи `trino_check` появится строка `Trino читает таблицу iceberg.sandbox.events: строк 20, снимков 2`
- все три задачи DAG завершатся в состоянии **Success**



Статус выполнения DAG

Результат подтверждает, что Spark записывает таблицу Iceberg в Yandex Object Storage и регистрирует её через общий Hive Metastore, а Trino находит эту же таблицу и читает её через каталог `iceberg`.

Два snapshots в Trino дополнительно подтверждают, что оба движка работают с одной историей состояний таблицы.

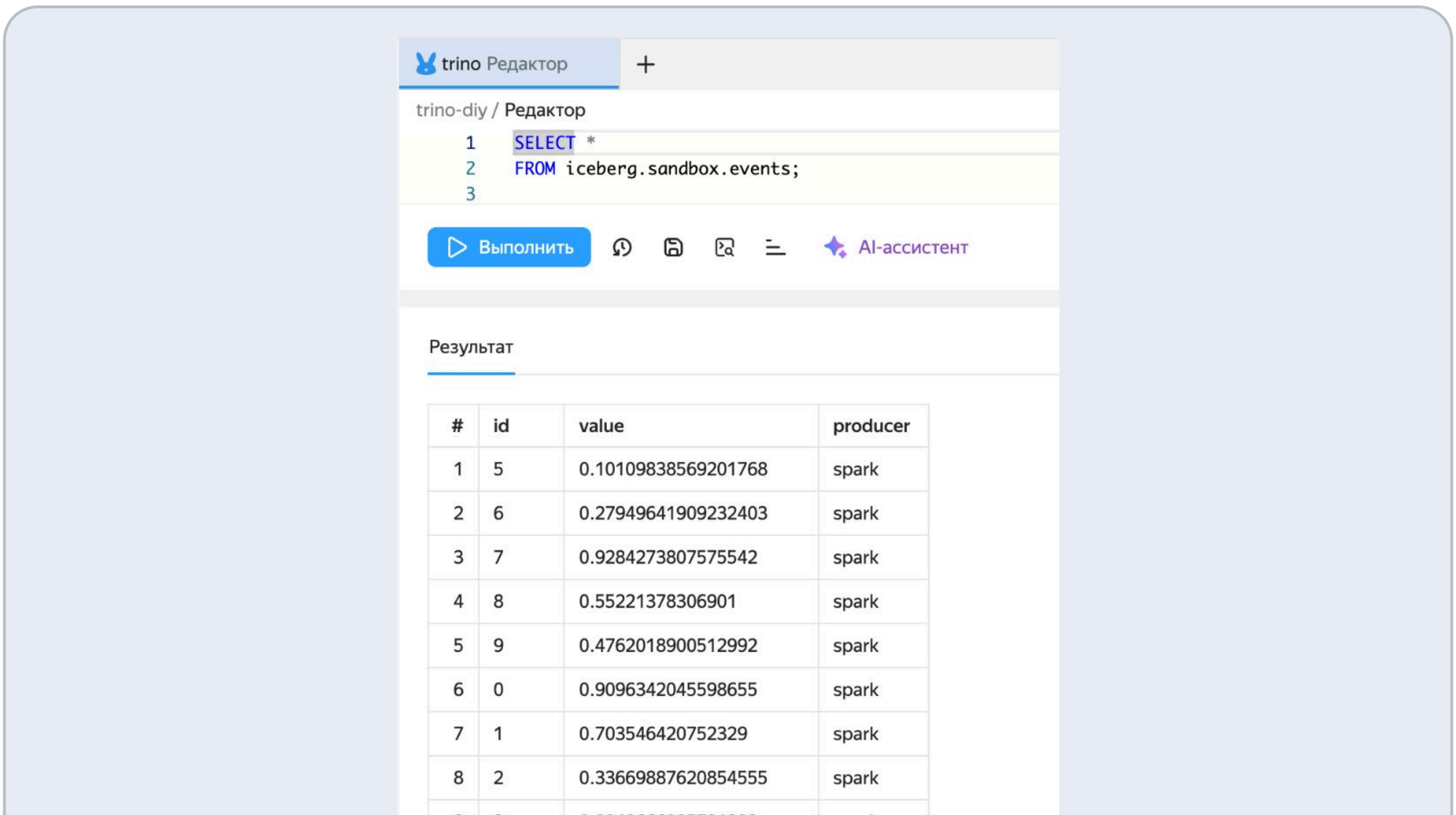
Проверка результата в Yandex WebSQL

После успешного выполнения DAG таблица `sandbox.events` остаётся в каталоге `iceberg`. Откройте Yandex WebSQL и убедитесь, что она доступна в схеме `sandbox`.

Выполните запрос:

```
SELECT *
FROM iceberg.sandbox.events;
```

В результате должно вернуться 20 строк, записанных Spark в ходе приёмочного теста.



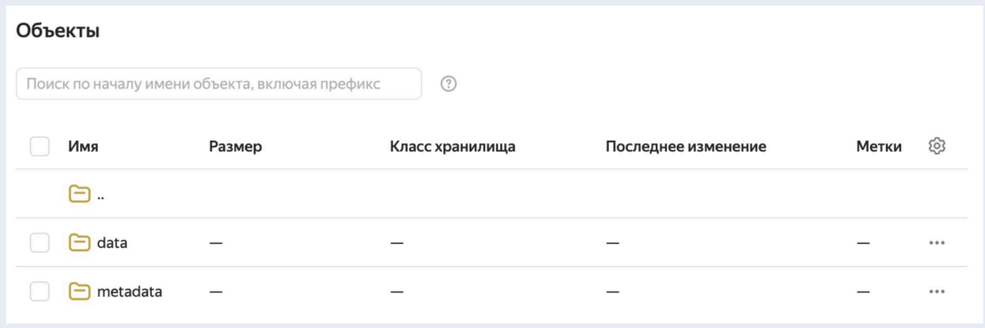
#	id	value	producer
1	5	0.10109838569201768	spark
2	6	0.27949641909232403	spark
3	7	0.9284273807575542	spark
4	8	0.55221378306901	spark
5	9	0.4762018900512992	spark
6	0	0.9096342045598655	spark
7	1	0.703546420752329	spark
8	2	0.33669887620854555	spark
9	3	0.9249666285721222	spark
10	4	0.4932977516591668	spark
11	10	0.19975660344812918	spark
12	11	0.6647450049331589	spark
13	12	0.21684985668883427	spark
14	13	0.6260361458490687	spark
15	14	0.7322939829803481	spark
16	15	0.13730280692314445	spark
17	16	0.29728872093816605	spark
18	17	0.9001096391053436	spark
19	18	0.1389787669151602	spark
20	19	0.714533751218562	spark

Результат проверки в Yandex WebSQL

Проверка результата в Yandex WebSQL

Теперь откройте бакет `lakehouse101` в Yandex Object Storage и перейдите по пути:

`internal/sandbox.db/events/`



Содержимое бакета lakehouse101

Внутри таблицы вы увидите два каталога:

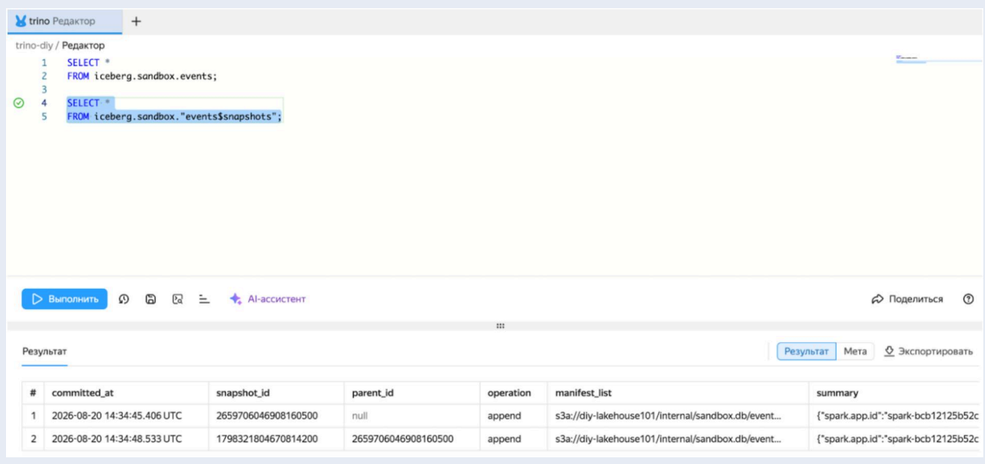
- `data` — Parquet-файлы с данными
- `metadata` — файлы метаданных Iceberg, которые описывают состояние и историю таблицы

Так таблица `sandbox.events`, доступная через SQL, физически представлена набором файлов в Yandex Object Storage.

Затем вернитесь в Yandex WebSQL и проверьте историю состояний таблицы:

```
SELECT *
FROM iceberg.sandbox."events$snapshots";
```

В таблице snapshots должны быть видны два снимка, созданные двумя последовательными операциями записи Spark.



Результат запроса — история состояния таблицы (snapshots)

Эта проверка подтверждает, что Trino читает не только текущие данные, но и метаданные Iceberg, созданные Spark через общий Hive Metastore.

Поздравляем — Lakehouse работает. Вы развернули основные компоненты контура и проверили их совместную работу сквозным тестом. Теперь этот контур можно использовать как основу для загрузки, обработки и аналитики данных.

Управление расходами

Если вы планируете продолжить работу с контуром позже, не удаляйте созданные ресурсы. Остановите кластеры Trino, Spark и Airflow, а также Metastore-сервер. Так вычислительная часть контура будет потреблять минимальный объём ресурсов, а данные и настройки сохранятся.

Yandex Object Storage продолжит хранить файлы таблиц и метаданные Iceberg. После повторного запуска сервисов можно продолжить работу с тем же Lakehouse-контуром.

Итоги и что делать дальше

Вы собрали работающий аналитический контур Lakehouse в Yandex Cloud:

- Сервисный аккаунт с ролями для взаимодействия компонентов
- Yandex Object Storage с отдельными бакетами для данных, PySpark-кода и DAG-файлов
- Hive Metastore как общий каталог метаданных
- Trino для SQL-запросов к таблицам Iceberg
- Spark для распределённой обработки данных
- Airflow для оркестрации заданий
- Сетевой доступ Airflow к Trino
- Сквозной приёмочный тест, который подтвердил совместную работу всех компонентов

В результате Spark создал таблицу `sandbox.events` в формате Iceberg, записал её файлы в Yandex Object Storage и зарегистрировал таблицу в Hive Metastore. Trino прочитал ту же таблицу через каталог `iceberg`, а Airflow связал оба действия в единый DAG.

Теперь этот контур можно использовать как основу для собственного пайплайна данных: подключить источник, организовать загрузку данных, добавить преобразования в Spark и аналитические запросы в Trino. Схему `sandbox` можно оставить для экспериментов, а структуру рабочих слоёв определить уже под конкретную задачу.