Yandex Cloud

Как обрабатывать данные с помощью Spark в облаке

Максим Зиналь
Архитектор продукта
Yandex Data Proc

Дмитрий Рыбалко
Архитектор продукта
Yandex DataSphere

Что мы понимаем под ML-разработкой и дата-инжинирингом?

Качественные ML-модели требуют большого количества подготовленных данных

- Apache Spark для сбора и подготовки данных
- Взаимодействие инструментов ML и Spark
- Совместная работа Data engineer и Data scientist / Аналитик

Кейсы: миграция в Yandex Cloud

Фирма «Вечная классика»

Миграция с локальной инфраструктуры

Компания «Модный тренд»

Миграция с облачного провайдера

Миграция с локальной инфраструктуры в облачную

Типовой локальный стек для обработки «больших данных»

Локальная инфраструктура:

- Собственный кластер
- HDFS (реже Ceph, GlusterFS)
- Spark
- JupyterHub

Особенности:

- Несколько команд в общем кластере
- Самостоятельная поддержка железа и ПО
- Медленное выделение ресурсов
- Сложная декомиссия

«Наивный» подход к миграции

1. Сделаем один большой кластер и всё туда скопируем!
2. Переключим процессы на новые ресурсы

«Наивный» подход к миграции

1. Сделаем один большой кластер и всё туда скопируем!
2. Переключим процессы на новые ресурсы

⊕ **Плюсы:**

- Минимальный объём работы
- Не надо никого переучивать

«Наивный» подход к миграции

1. Сделаем один большой кластер и всё туда скопируем!
2. Переключим процессы на новые ресурсы

Плюсы:

- Минимальный объём работы
- Не надо никого переучивать

Минусы:

- Стоимость ресурсов
- Виртуальные машины vs Физические серверы
- Режим «сделай сам»
- А зачем тогда мигрировали?

Оптимизация для варианта в облаке

Ресурсы on-demand
в облаке

- Плата только
  за потребляемые ресурсы
- Легко отказаться
  от «лишних» мощностей
  или добавить недостающие

Оптимизация для варианта в облаке

Ресурсы on-demand в облаке

- Плата только за потребляемые ресурсы
- Легко отказаться от «лишних» мощностей или добавить недостающие

Кластеры «под команду» и/или временные кластеры

- Общие данные в Object Storage
- Отдельно регламентные и интерактивные операции
- Унификация «начинки» кластеров

Оптимизация для варианта в облаке

Ресурсы on-demand в облаке

- Плата только за потребляемые ресурсы
- Легко отказаться от «лишних» мощностей или добавить недостающие

Кластеры «под команду» и/или временные кластеры

- Общие данные в Object Storage
- Отдельно регламентные и интерактивные операции
- Унификация «начинки» кластеров

Ограничение доступа на сетевом уровне

Управляемые сервисы и экосистема

Управляемые сервисы

- Меньше нагрузка на DevOps-инженеров
- Инструменты автоматизации (Terraform, Airflow)
- Логи и мониторинг «из коробки»

Экосистема

- Смежные сервисы — базы данных, Kafka, …
- Межсервисные интеграции

Object Storage для хранения данных

1. Экономичное хранение
2. Надёжность
3. Простота
4. Автоматическое масштабирование
5. Удобная интеграция

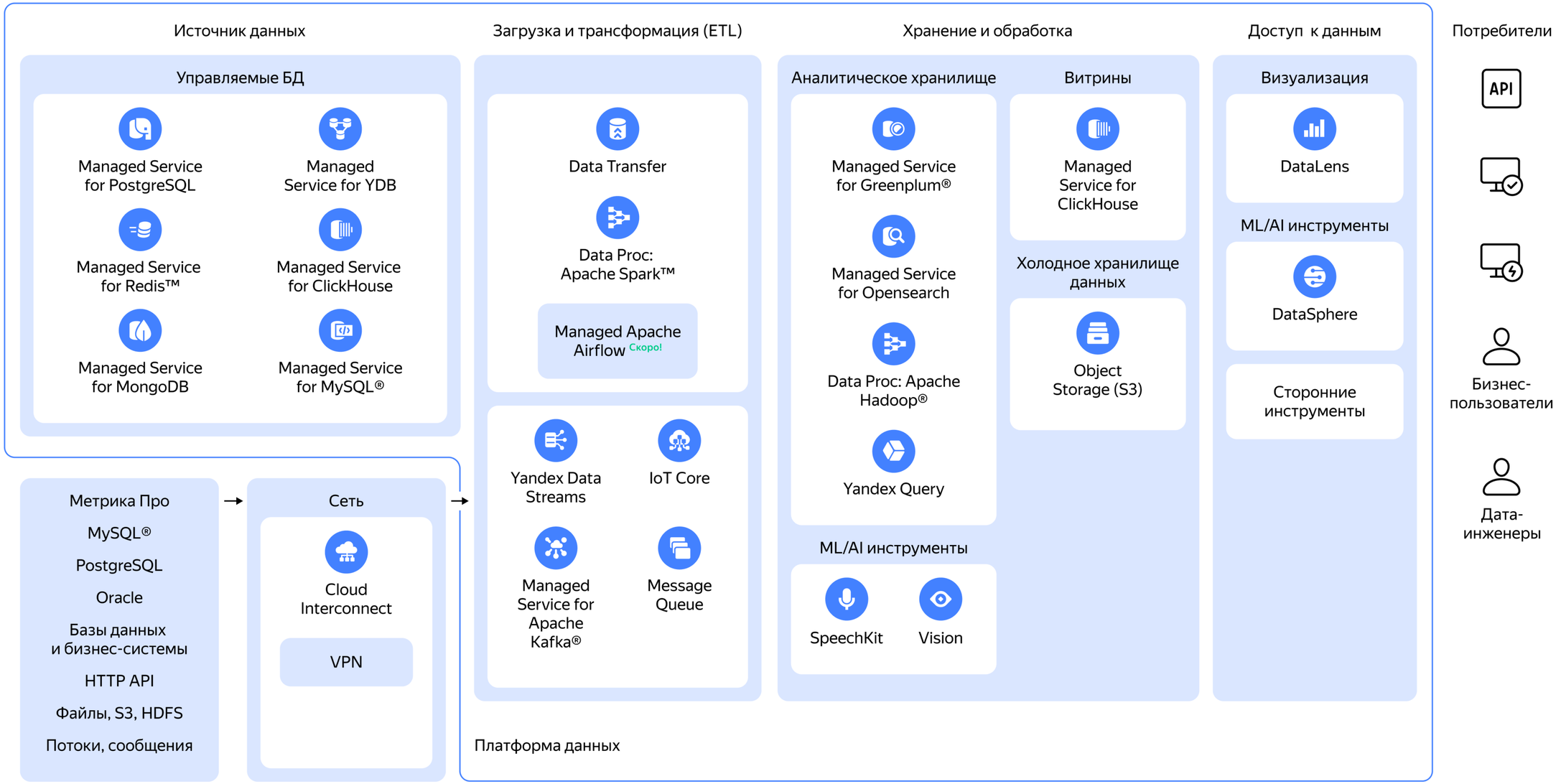
Централизованный Hive Metastore для метаданных

- Структура таблиц
- Размещение данных
- Формат хранения
- Схема партиционирования

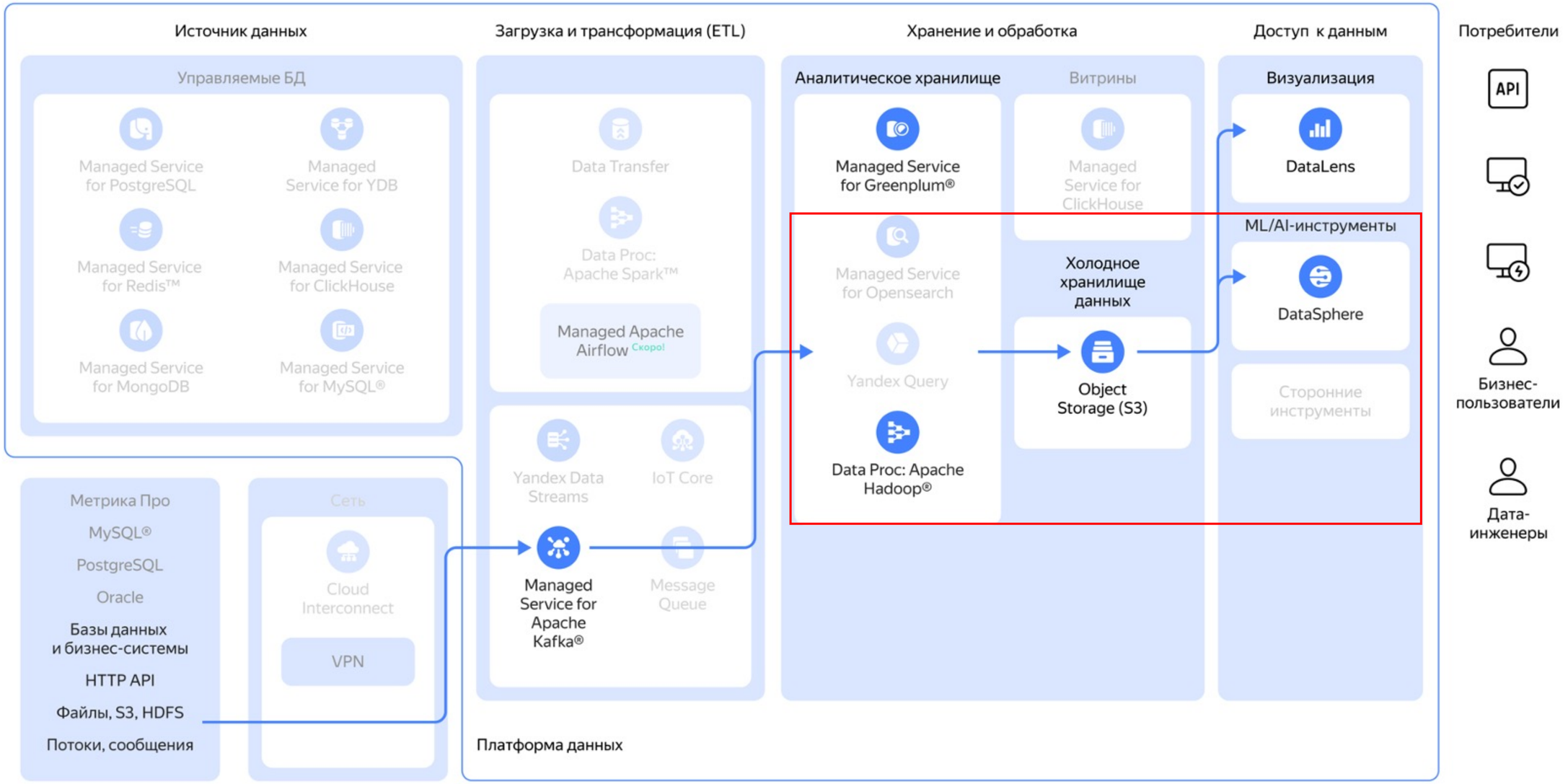
Object Storage для хранения данных

1. Экономичное хранение
2. Надёжность
3. Простота
4. Автоматическое масштабирование
5. Удобная интеграция
6. Нет переименования файлов
7. Неконсистентные листинги
8. Весь доступ через сеть

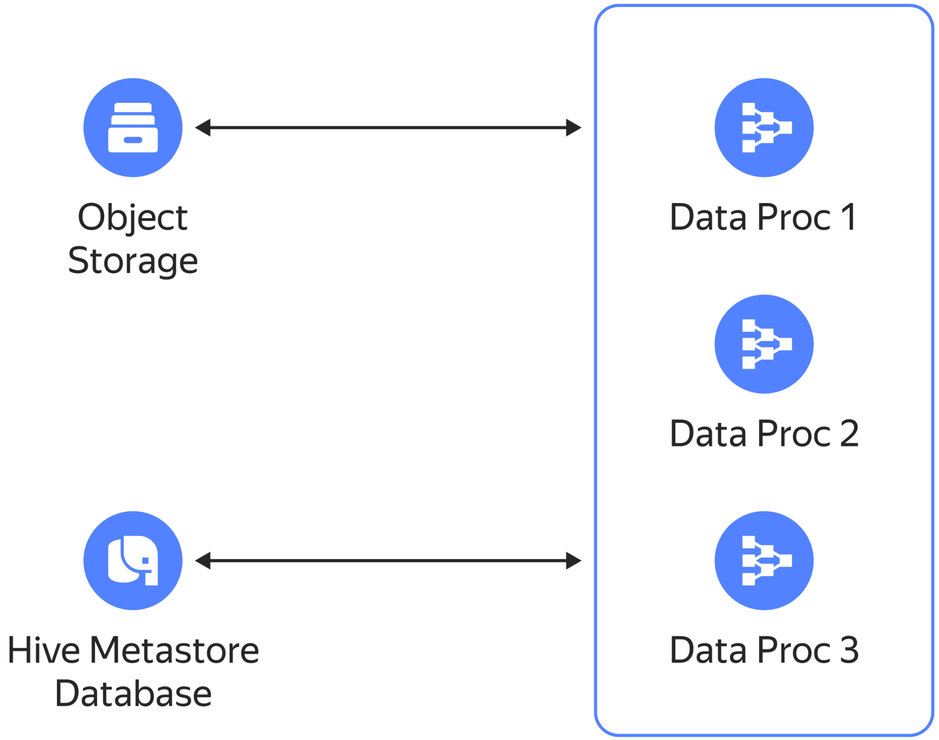
Платформа данных Yandex Cloud

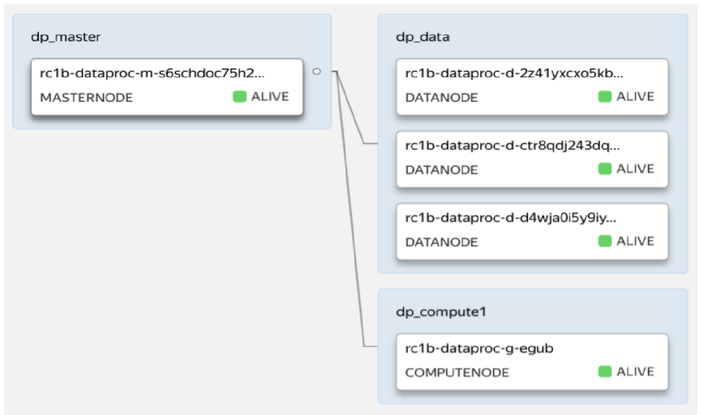


Платформа данных Yandex Cloud



Data Proc как управляемый сервис Spark





Apache Airflow

или другой оркестратор

```
yc dataproc cluster create ...
yc dataproc job create-spark ...
yc dataproc job get ...
yc dataproc job log ...
```

Автомасштабирование Data Proc

1

- **Добавление узлов: очереди YARN**
- Pending containers count

2

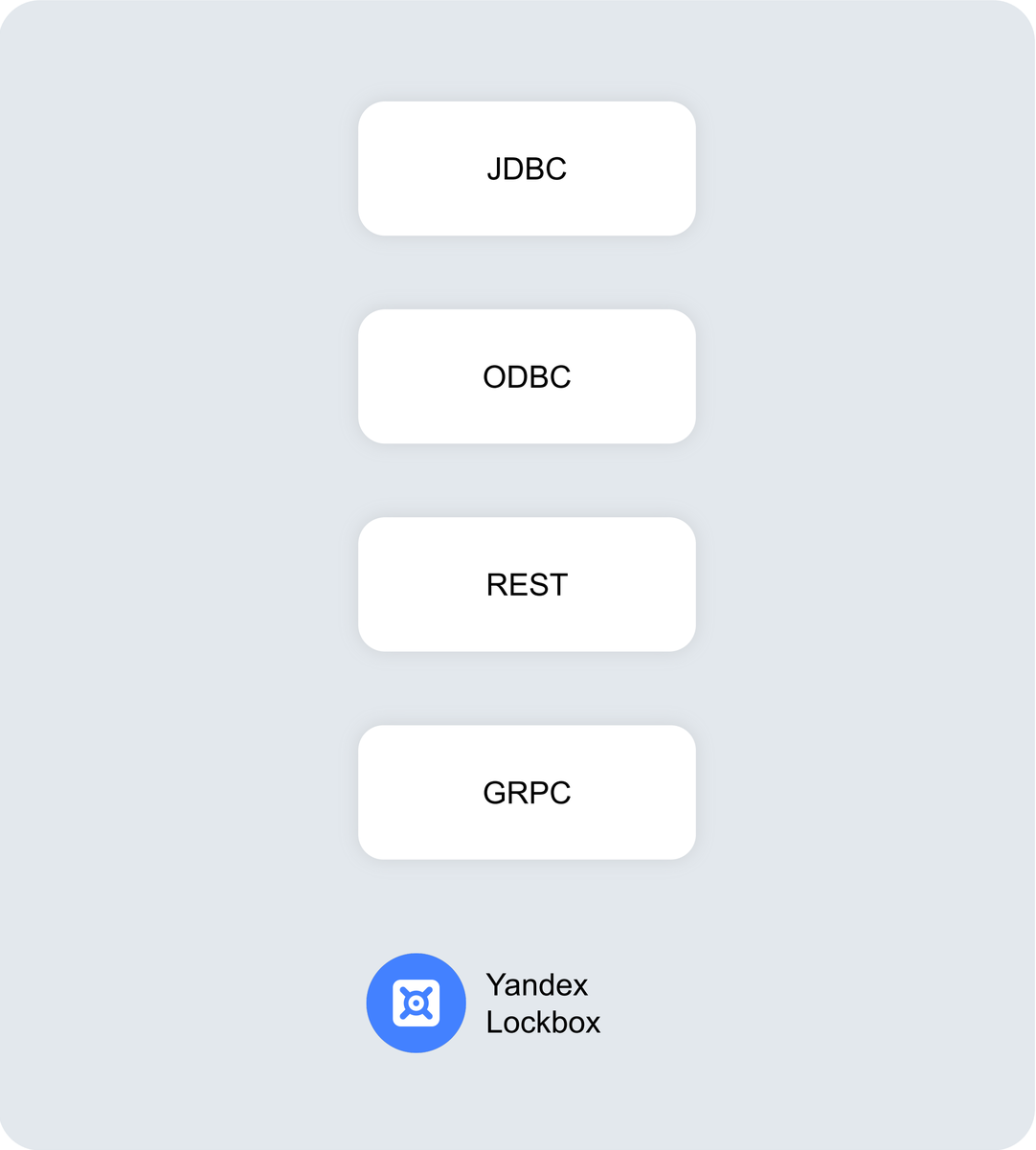
- **Удаление узлов: выполняемые контейнеры YARN**

3

- **Рестарт контейнеров при удалении**
- Часть узлов будет остановлена
- Не устанавливайте spark.task.maxFailures=1

Работа с секретами

1. Object Storage «из коробки»
2. Внешние источники данных: ключи, логины, пароли
3. Сервис Yandex Lockbox
   - Хранение секретов
   - Контроль доступа
   - API для безопасного получения секретов



DataSphere: интерфейс к Spark и среда разработки моделей

Гибкость и масштабируемость

1. Привычная среда JupyterLab
2. Самостоятельное управление конфигурациями железа
3. Командная работа: обмен ресурсами и артефактами
   - Docker-образы
   - Секреты
   - Подключения к S3
4. Интеграция с Git
5. Обмен кодом между локальной и облачной средами

Переключение между конфигурациями

1. Возможность гибко переключаться между конфигурациями GPU NVIDIA V100 и A100
2. Ограничение используемых ресурсов и трат в проекте и сообществе:
   - Ограничение по балансу
   - Ограничение набора доступных конфигураций
   - Ограничения на создания ресурсов

Конфигурация
c1.4 (4 vCPU, 0 GPU)
c1.8 (8 vCPU, 0 GPU)
c1.32 (32 vCPU, 0 GPU)
c1.80 (80 vCPU, 0 GPU)
g1.1 (8 vCPU, 1 GPU V100)
g1.2 (16 vCPU, 2 GPU V100)
g1.4 (32 vCPU, 4 GPU V100)
g2.mig (4 vCPU, 1/8 GPU A100)
g2.1 (28 vCPU, 1 GPU A100)
g2.2 (56 vCPU, 2 GPU A100)
g2.4 (112 vCPU, 4 GPU A100)
g2.8 (224 vCPU, 8 GPU A100)

Основная особенность – гибкое управление конфигурациями

Режим Dedicated

- Запуск каждого Jupyter Notebook на выделенной виртуальной машины (ВМ)
- Единая конфигурация для всех ячеек ноутбука
- Самостоятельный запуск и остановка ВМ
- Возможность автоматического автоотключения ВМ при неактивности

Режим Serverless

- ВМ выделяется автоматически под запуск каждой ячейки
- Регулярная сериализация состояния ноутбука, создание контрольных точек
- Гибкое перемещение между конфигурациями железа внутри одного ноутбука

Пример запуска вычислений

Для запуска на кластере Spark в ячейке необходимо указать id кластера и сессию.
Там же могут быть указаны переменные, в которые необходимо сохранить результат

[ ]:
```
#!spark --cluster {cluster-name} --return_variables df
df = spark.sql("SELECT * FROM test;")
```

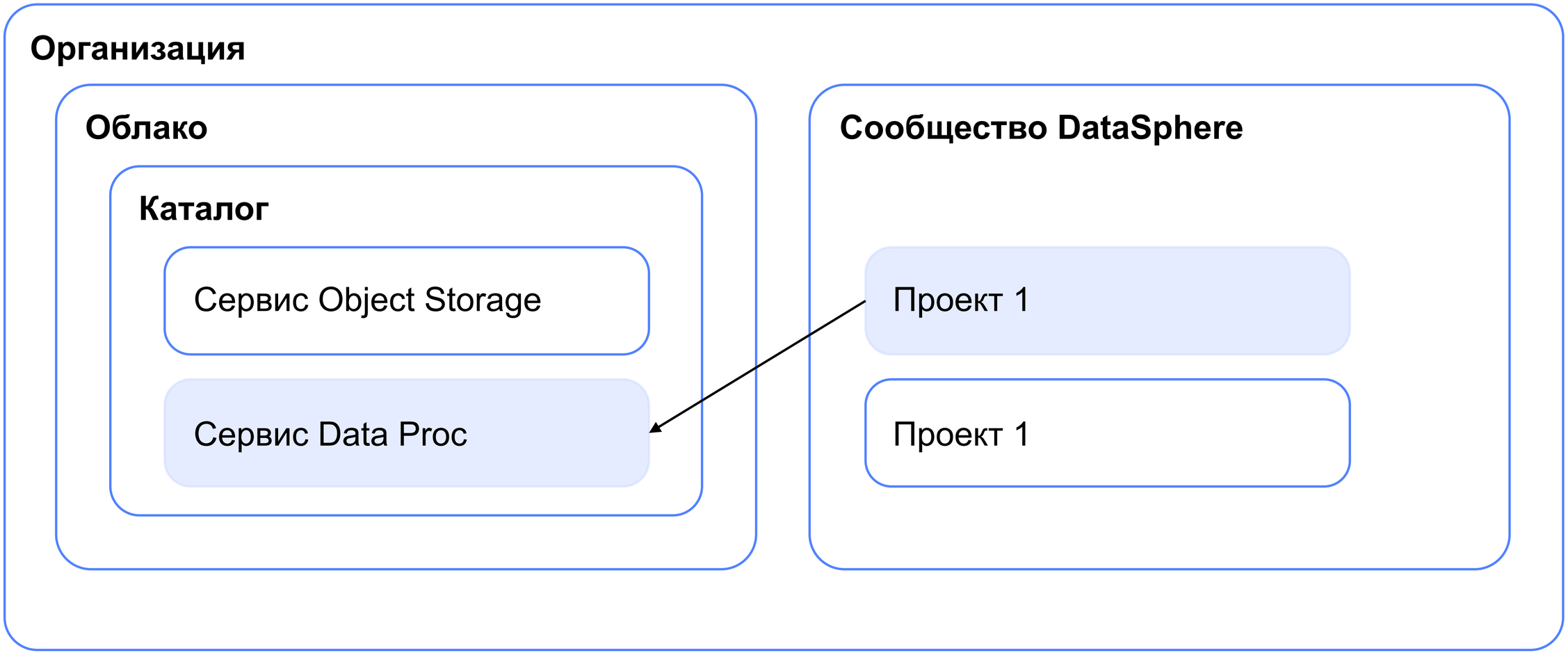
[ ]:
```
%create_livy_session \
--cluster dataproc-test-cluster \
--id ses1 \
--conf spark.jars.packages=io.delta:delta-core_2.12:0.8.0 \
--conf spark.sql.extensions=io.delta.sql.DeltaSparkSessionExtension \
--conf spark.sql.hive.metastore.sharedPrefixes=com.amazonaws,ru.yandex.cloud \
```

[ ]:
```
#!spark --cluster dataproc-for-datasphere --session ses2
spark.sql("CREATE TABLE students (name VARCHAR(64), address VARCHAR(64), student_id INT);")
```

Взаимодействие между DataSphere и Data Proc

- Запуск заданий Spark из DataSphere через Livy (REST)
- Передача переменных из DataSphere в Data Proc
- Файловый обмен: magic-команды или Object Storage

Организация аутентификации между сервисами



Шаблоны Data Proc

Позволяют быстро развернуть временные кластера Data Proc внутри проектов DataSphere.

- Шаблон определяет конфигурацию кластера Data Proc
- Автоматически удаляется при отсутствии вычислений более 2 часов

Тип кластера	Количество хостов	Объем дисков
XS	1	384 ГБ HDD
S	4	1152 ГБ SSD
M	8	2176 ГБ SSD
L	16	4224 ГБ SSD
XL	32	8320 ГБ SSD

Миграция с облачного провайдера

Составные части процесса миграции

Данные

- Подключения к источникам
- Аналитические базы данных
- Архивы

Составные части процесса миграции

Данные

- Подключения к источникам
- Аналитические базы данных
- Архивы

Процессы обработки

- Состав инструментов
- Версии, совместимость
- Оптимизация для облачной среды

Составные части процесса миграции

Данные

- Подключения к источникам
- Аналитические базы данных
- Архивы

Процессы обработки

- Состав инструментов
- Версии, совместимость
- Оптимизация для облачной среды

Анализ и визуализация

- ML, BI инструменты
- Свой стек либо управляемые сервисы
- Публикация обученных моделей

Другой облачный провайдер: сходства

- Object Storag
- Apache Spark
- Delta Lake

- Notebooks → DataSphere
- Auto Loader → триггеры Cloud Functions
- Delta LiveTables → Spark Structured Streaming
- Jobs → триггеры Cloud Functions или Apache Airflow
- DBFS → GeeseFS

«Наивный» подход к миграции

Основные действия:

- Скопируем данные и ноутбуки
- Настроим заново процессы

В этот раз должен сработать!

«Наивный» подход к миграции

Основные действия:

- Скопируем данные и ноутбуки
- Настроим заново процессы

Дополнительно потребуется:

- Планирование ресурсов
- Разработка интеграций
- Тестирование производительности
- Оптимизация запросов

Основная особенность — гибкое управление конфигурациями

У заданий Spark много параметров

- `spark.{driver,executor}.{cores,memory}`
- `spark.jars[.packages]`
- `spark.serializer`
- `spark.scheduler.mode`
- `spark.task.maxFailures`
- `spark.dynamicAllocation.enabled`
- `...`

Неверно выбранные настройки:

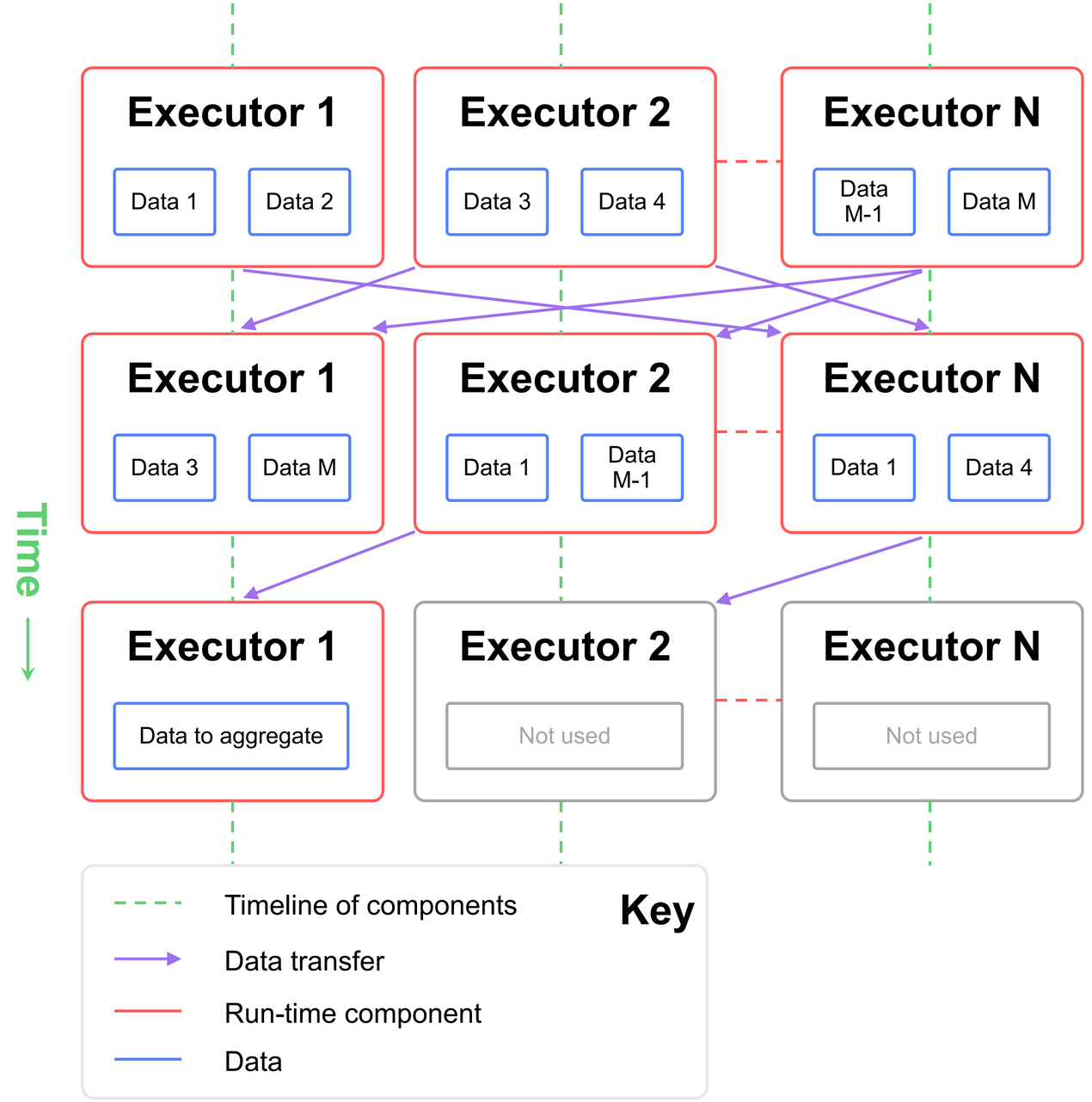
- **Замедляют работу**
- **Приводят к ошибкам и остановке**
- `java.lang.OutOfMemoryError: Java heap space`
- `java.lang.OutOfMemoryError: GC overhead limit exceeded`

Оптимизация заданий

Применима «обычная» экспертиза Hadoop и Spark

Существует порог входа

- Распределённые вычисления
- Администрирование YARN/HDFS/…
- Разработка Spark
- Оптимизация Spark SQL, Hive SQL



Миграция возможна!

Но в каждом случае требует индивидуального подхода и определенного набора работ

Спасибо!

Подробная документация с кейсами и примерами, которая поможет вам найти решение вашего вопроса

clck.ru/nzctC

Максим Зиналь
Архитектор продукта
Yandex Data Proc
mzinal@yandex-team.ru

Дмитрий Рыбалко
Архитектор продукта
Yandex DataSphere
dm-rybalko@yandex-team.ru