федеральное государственное автономное образовательное
учреждение высшего образования

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»

ОТЧЕТ

по Лабораторной работе №4

«Финальное решение с реализацией»

по дисциплине «**Облачные и туманные вычисления**»

Авторы: Кулаков Н.В. P34312

Лысенко Д.С. P34121

Факультет: ПИиКТ

Преподаватель: Перл О.В.

Санкт-Петербург, 2023

Содержание

Решение

Будет разработано веб-приложение с простым интерфейсом, в котором пользователь сможет создавать и использовать уже существующие пайплайны, шаги которого последовательно обрабатываются нейросетевыми моделями.

- Решение было изменено на API с заявленным функционалом, как изначально обсуждалось на сдаче 1 ЛР. В качестве демонстрации работоспособности и результатов инференций было написано web-приложение, которое использует ограниченную часть API.

Используемые сервисы

1. Virtual Machine. Нейросетевые модели для инференции (изображений и текстов). Для генерации различных типов ответов будут созданы отдельные нейросетевые сервисы.

1.2. Virtual Machine. Сервер для обработки запросов пользователей, также осуществляющий сохранение результатов генерации и метаданных (backend + frontend).

2. YDB для хранения информации о пользователях, метаданные о сгенерированных записях и ссылки на бакеты/данные в S3. - был заменен на Managed Service for PostgreSQL, поскольку в библиотеке Yandex SDK для Java не реализована функциональность JPA.

3. Yandex Object Storage для хранения текста и изображений.

4. TLS Certificate Manager. Сервис для управления TLS сертификатами для домена (фронта).

5. Yandex Message Queue для управления инференцией (процесс генерации) и распределения задач между нейросетевыми нодами, генерации очереди пользователей.

6. Yandex Monitoring для мониторинга основных метрик сервиса, например, средняя скорость инференции, использование оперативной памяти и количество сгенерированных изображений и так далее.

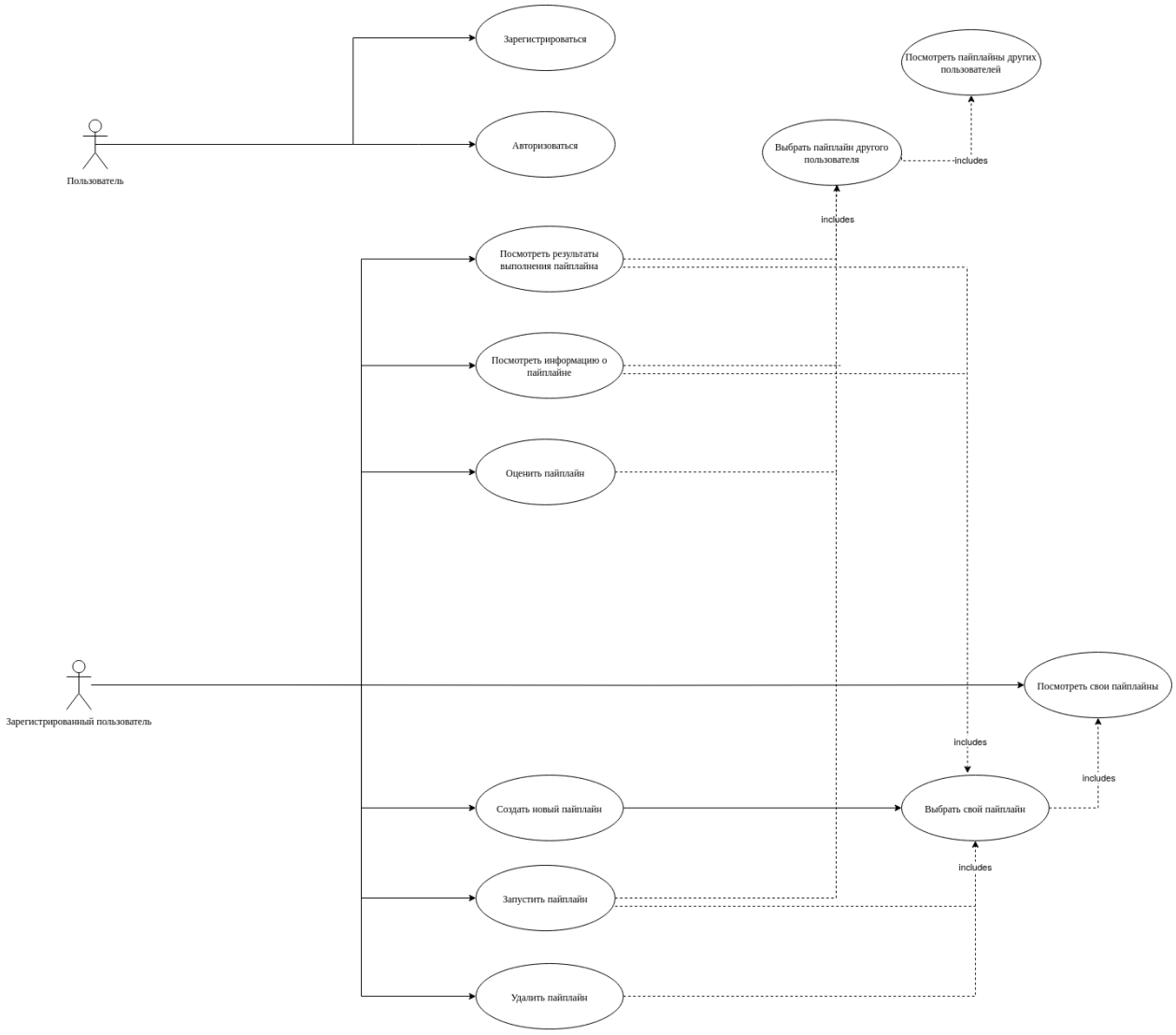
В качестве облачных провайдеров были выбраны Yandex Cloud и Selectel (для машин с GPU)

Функциональные требования

1. Приложение должно предоставлять возможность авторизации и регистрации пользователей
2. Приложение должно предоставлять возможность просматривать существующие пайплайны пользователя, либо других пользователей
3. Приложение должно предоставлять возможность создавать новый пайплайн
4. Приложение должно предоставлять возможность удалять уже доступный пайплайн
5. Приложение должно предоставлять возможность просмотра оценки пайплайна
6. Приложение должно предоставлять возможность ставить оценку пайплайну другого пользователя
7. Приложение должно предоставлять возможность выбирать любой доступный в системе пайплайн для его запуска
8. Приложение должно предоставлять возможность выбора API инференции при использовании пайплайна
    a. Приложение должно предоставлять доступ к следующим API:
        i. Собственное API (инференция на виртуальной машине с GPU)
9. Приложение должно предоставлять возможность запуска пайплайна
10. Приложение должно предоставлять возможность сохранения результатов пайплайна на сервере
11. Приложение должно предоставлять возможность просмотра результатов пайплайна на сервере
12. Приложение должно предоставлять возможность удаления результатов пайплайна на сервере

Use-case diagram

После определенных размышлений был запрещен доступ неавторизованным пользователям к пайплайнам. В связи с этим изменилась Use-Case диаграмма:

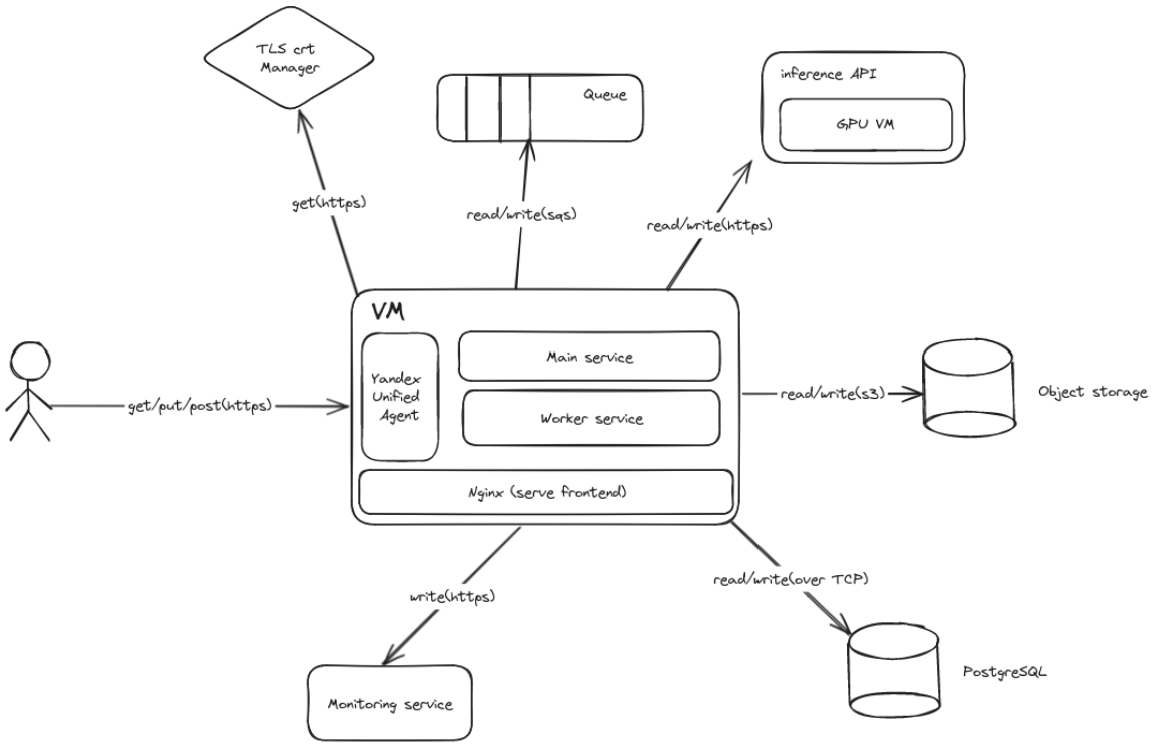


Роли

В системе существует только одна роль - зарегистрированный пользователь. Доступ к API неавторизованному пользователю, за исключением регистрации и авторизации, запрещен.

Примерная архитектура решения

Измененная архитектура решения:



Бэкенд был разнесен на 2 сервиса (Main - взаимодействующий с клиентами, Worker - обслуживающий провайдеры с нейронными сетями), общающихся через Queue. Первоначально разными участниками проектами виделось по-разному (реализовывать в виде отдельных модулей, либо сервисов), в связи с этим случилась рассинхронизация между архитектурой и диаграммой развертывания.

TLS Manager - с помощью этого сервиса будет генерироваться Let’s Encrypt сертификат по нашему домену, на котором будет висеть наше приложение. Автоматически будет осуществляться продление сертификата при подходе к окончанию срока действия, на стороне виртуалки предполагается, что при наступлении этого события будет выгружаться сертификат из менеджера и замещать старый.

Monitoring Service - на backend-е будет создан эндпоинт метрик, затем Yandex Unified Agent будет отправлять метрики в сервис мониторинга. При помощи заранее настроенных дешбордов можно будет посмотреть на визуализированные метрики по параметрам.

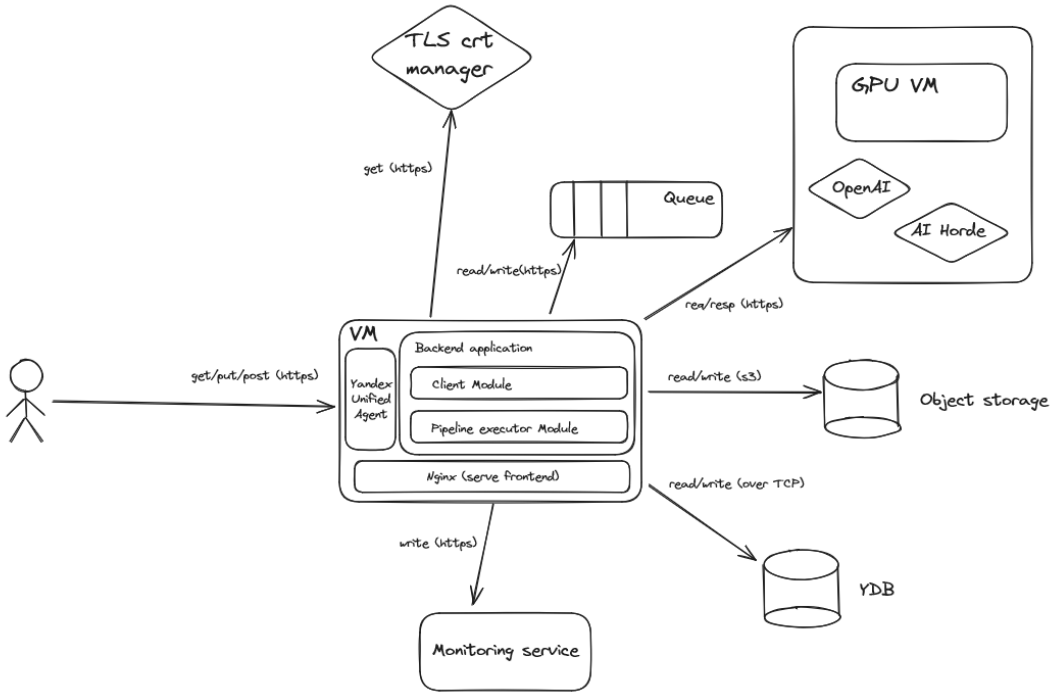
Queue - используется как очередь pipeline-ов на выполнение. От пользователя приходит запрос на выполнение пайплайна, который помещается в нее. Эти задачи считываются

отдельным модулем backend-а, который осуществляется последовательное делегирование этапов пайплайна на GPU или внешнее API.

Object Storage - используется для хранения файлов пайплайнов (json-ы), результатов выполнения пайплайнов (изображения или текст).

YDB - для хранения информации о пользователях, всех созданных пайплайнов (ссылок на Object Storage), а так же результатов выполнения пайплайнов (ссылок).
- был заменен на PostgreSQL по причине, описанной выше.

Первоначальная архитектура:



Стек технологий

- Main service: Java SE 17, Spring Boot, AWS SQS, AWS S3, Spring JPA (PostgreSQL), OpenAPI, Micrometer (Prometheus)
- Worker service: Java SE 17, Spring Boot, JMS (over AWS SQS), AWS S3, Micrometer (Prometheus)

Диаграмма развертывания

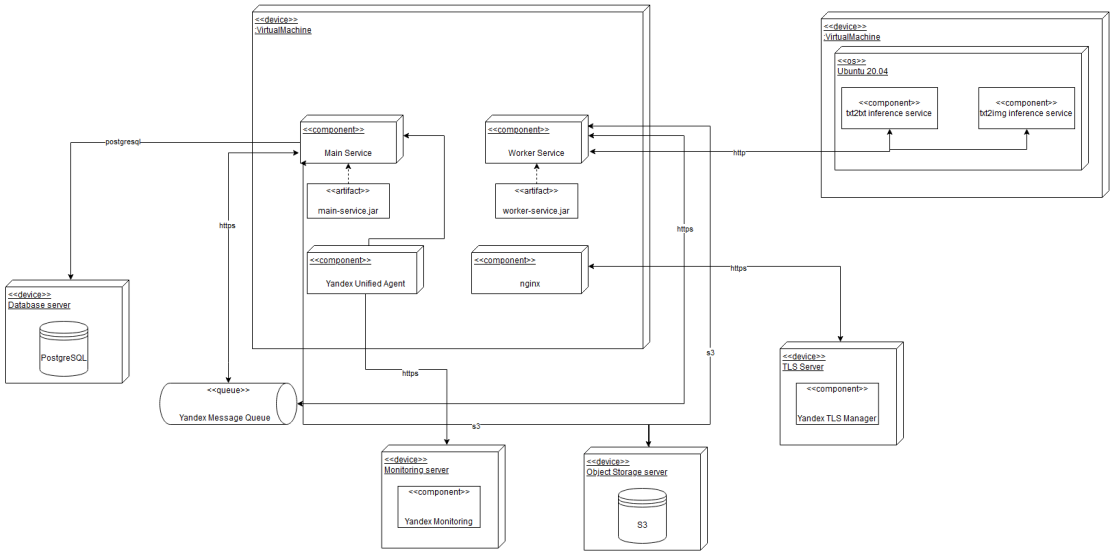
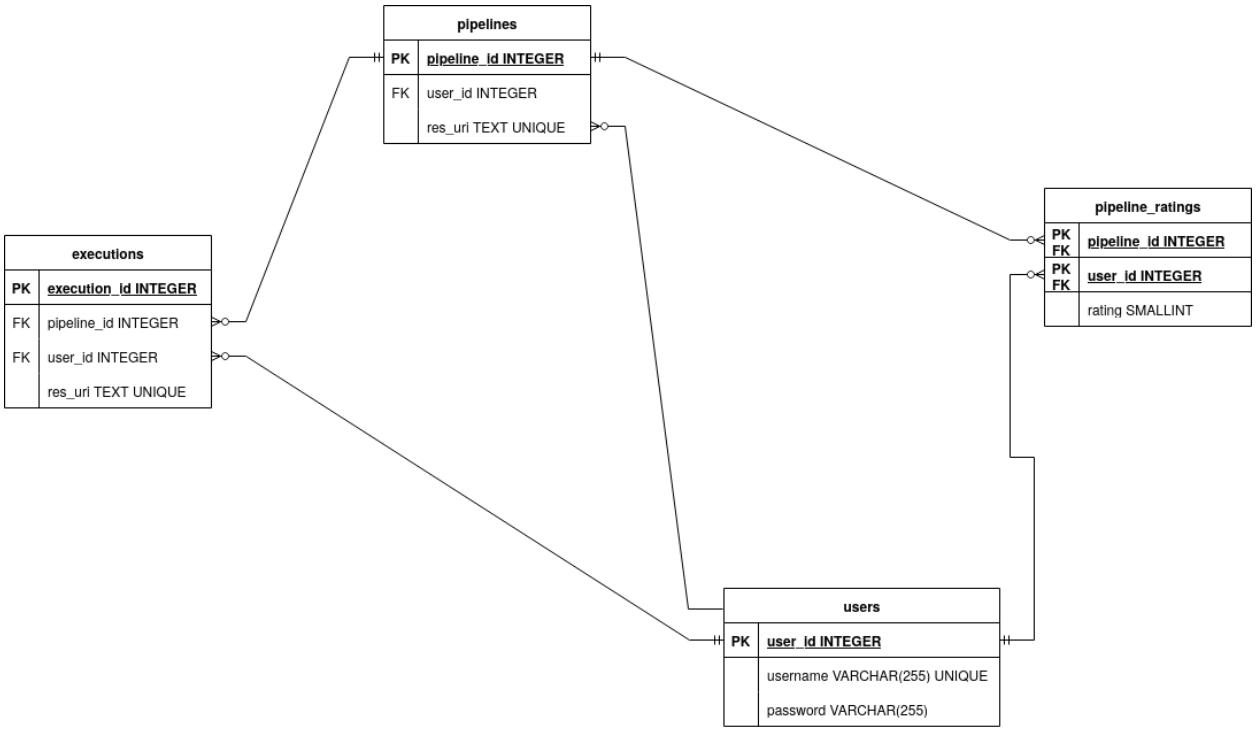


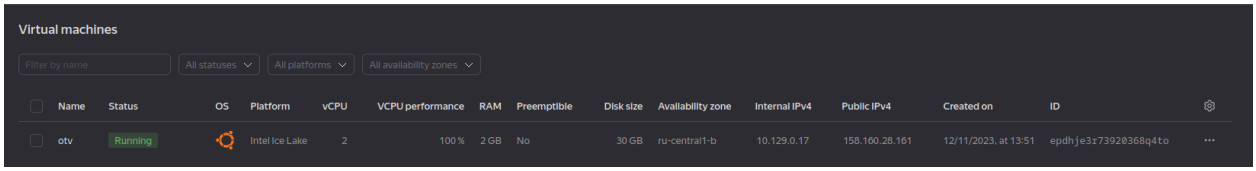
Диаграмма развертывания претерпела незначительные изменения - добавили взаимодействие между Main Service и S3 (забыли указать в третьем этапе), убрали сбор метрик с worker service (посчитали избыточным), передумали разворачивать сервисы в Docker (после консультаций с преподавателем)

Схема базы данных

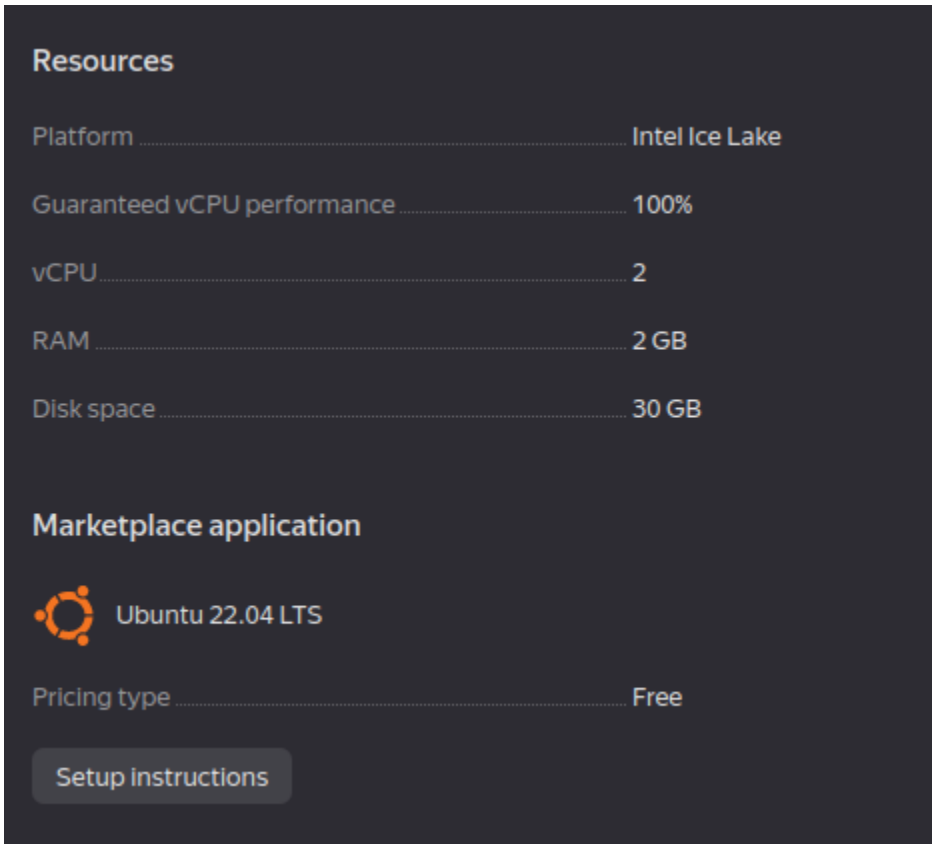


Virtual Machines

Была поднята виртуальная машина в Yandex Cloud:



Используемые ресурсы и установленная ОС:



В качестве хранилища данных был выбран HDD (дешево + пространство не будет использоваться для хранения данных приложения)
Ubuntu 22.04 была выбрана как самый распространенный дистрибутив Linux (редкие выпуски feature-обновлений, постоянные security обновления)

Были установлены nginx для обслуживания статического контента и бекэнда, Eclipse Temurin JDK 17 для запуска наших сервисов.

Пример системного юнита systemd для запуска Main Service:

```
[Unit]
Description=OTV Backend Service

[Service]
WorkingDirectory=/opt/backend-service
ExecStart=java -Xms128m -Xmx256m -jar -Dspring.profiles.active=prod backend.jar
User=otv
Type=simple
Restart=on-failure
RestartSec=10
StandardOutput=journal
StandardError=journal

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

Пример конфигурации веб-сервера:

```
upstream backend-service {
        server localhost:8080;
        keepalive 64;
}

server {
        listen 80 default_server;
        server_name spynad.ru www.spynad.ru;

        error_log /var/log/nginx/backend.error.log;
        access_log /var/log/nginx/backend.access.log;

        return 308 https://$server_name$request_uri;
}

server {
        listen 443 ssl http2 default_server;
        server_name spynad.ru www.spynad.ru;
        error_log /var/log/nginx/frontend.error.log;
        access_log /var/log/nginx/frontend.access.log;
        ssl_certificate /etc/nginx/ssl/chain.pem;
        ssl_certificate_key /etc/nginx/ssl/priv.pem;

        location / {
                root /opt/frontend/dist;
        }
```

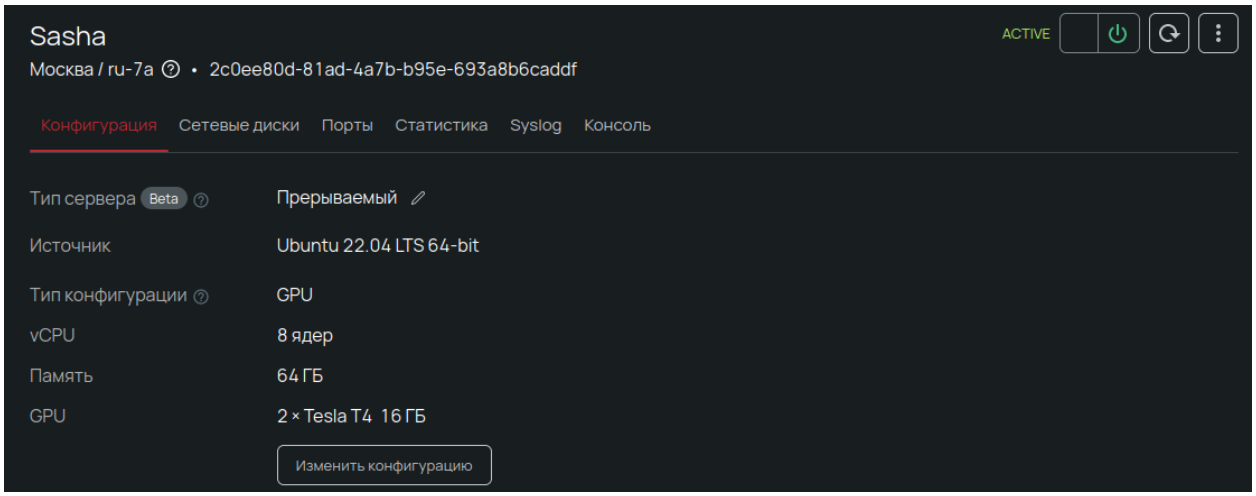
```
}

server {
        listen 443 ssl http2;
        server_name backend.spynad.ru www.backend.spynad.ru;

        error_log /var/log/nginx/backend.error.log;
        access_log /var/log/nginx/backend.access.log;
        ssl_certificate /etc/nginx/ssl/chain.pem;
        ssl_certificate_key /etc/nginx/ssl/priv.pem;

        location / {
                proxy_read_timeout 600s;
                proxy_send_timeout 600s;
                proxy_set_header X-Real-IP $remote_addr;
                proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
                proxy_ssl_server_name on;
                proxy_pass http://backend-service;
        }
}
```

Вторая виртуальная машина была поднята в Selectel.



Были установлен CUDA Toolkit (драйвера Nvidia, необходимые зависимости для выполнения вычислений на GPU), скомпилирован приложение инференции KoboldCPP и установлен stable-diffusion-webui.

Object Storage, описание пайплайна

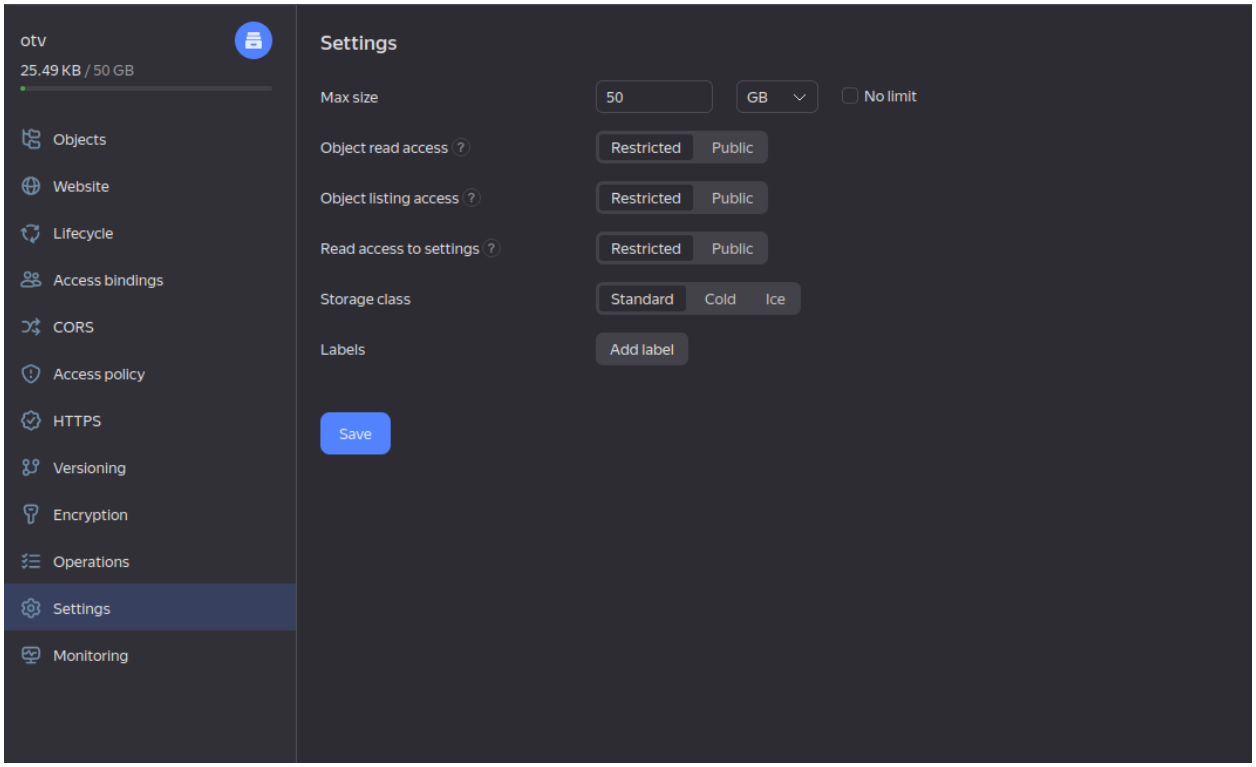
Пайплайны будут храниться в Object Storage в виде файла json.

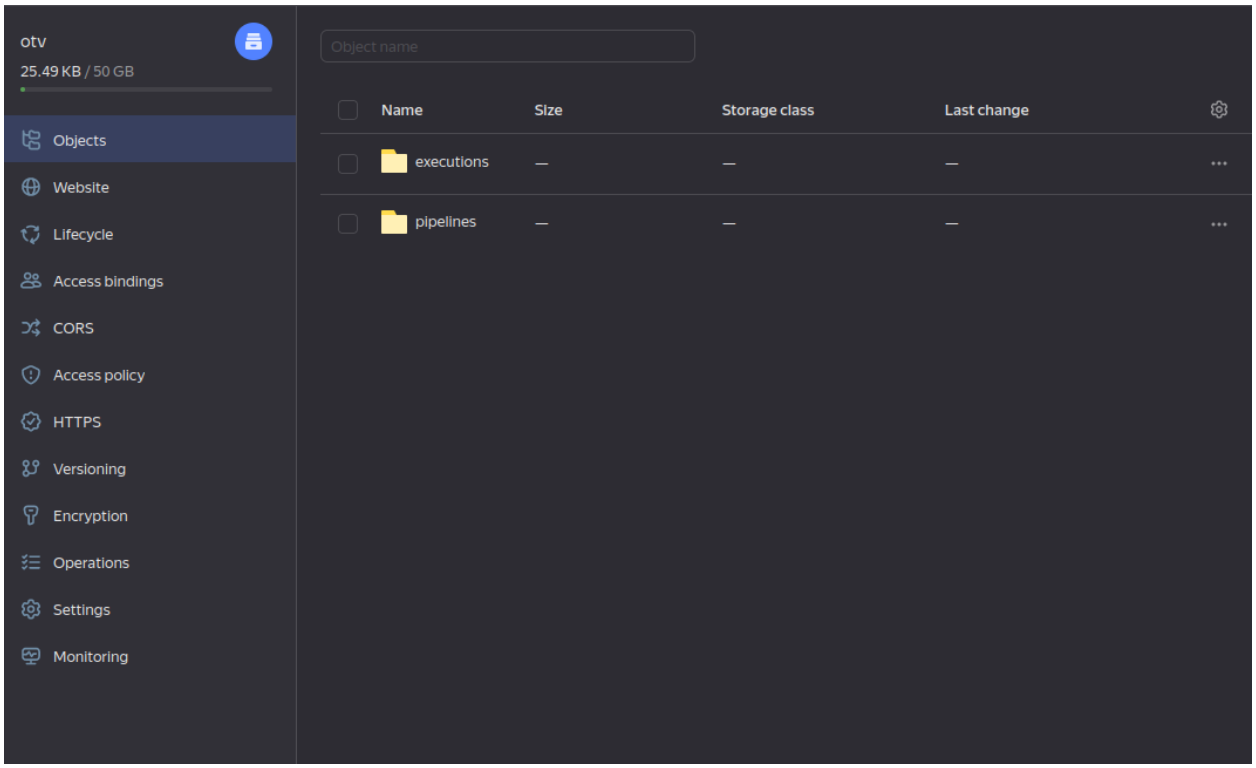
```
{
      "pipelineId": integer
      "name": string,
      "description": string,
      "author": string,
      "stages": [
            {
                  "type": string,
                  "inline": boolean,
                  "body": string,
                  "parameters": []
            },
            {
                  "type": string,
                  "inline": boolean,
                  "body": string,
                  "parameters": []
            }
      ]
}
```

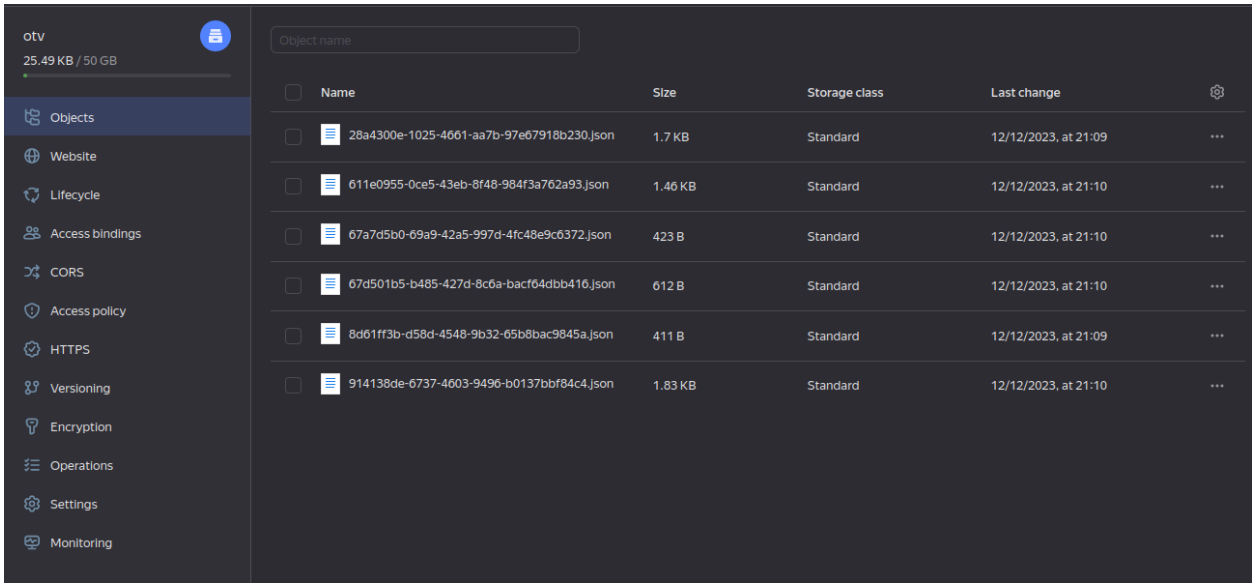
Сообщения, передаваемые в очередь выполнения запросов - совпадают с определением пайплайна.

- stages - этапы пайплайна.
- stages.type - тип задачи. Например, txt2img (генерация на основании изображения текста), txt2txt (генерация текста на основании текста).
- stages.inline - используется ли внутри пайплайна prompt, или ссылка на объект Object Storage. Это необходимо, поскольку Yandex Queue принимает максимально 256 KB в качестве записи, что требует использования ссылок, а не самих данных.
- stages.body - тело запроса (prompt), либо ссылка на само тело.
- stages.parameters в этапах пайплайна - различные конфигурационные параметры, которые непосредственно влияют на инференцию. Например, это может быть тип выборщика, используемый при генерации токенов в текстовых моделях, температура (случайный фактор) и т.д.

Было добавлено поле description - для описания пайплайна.

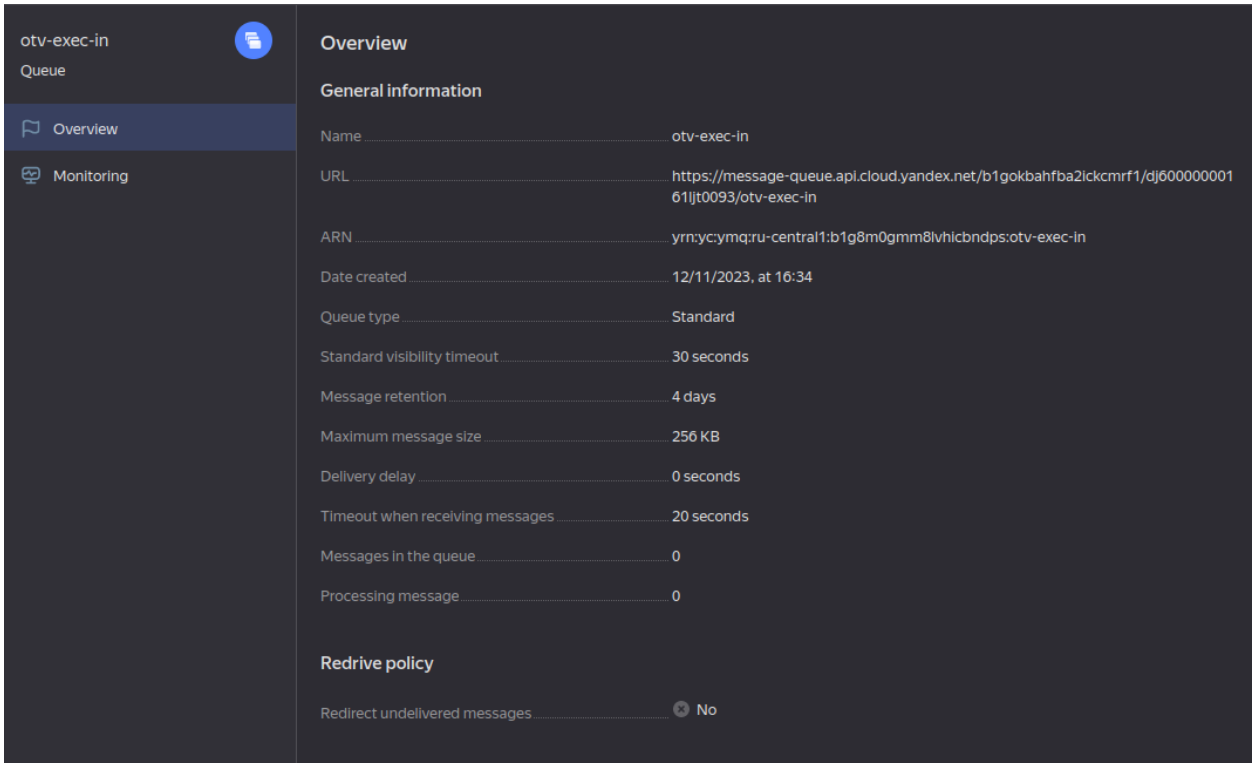

otv
25.49 KB / 50 GB
Objects
Website
Lifecycle
Access bindings
CORS
Access policy
HTTPS
Versioning
Encryption
Operations
Settings
Monitoring
Settings
Max size
50
GB
No limit
Object read access
Restricted
Public
Object listing access
Restricted
Public
Read access to settings
Restricted
Public
Storage class
Standard
Cold
Ice
Labels
Add label
Save


otv
25.49 KB / 50 GB
Objects
Website
Lifecycle
Access bindings
CORS
Access policy
HTTPS
Versioning
Encryption
Operations
Settings
Monitoring
Object name
Name
Size
Storage class
Last change
executions
pipelines



Message Queue, описание очередей и сообщений

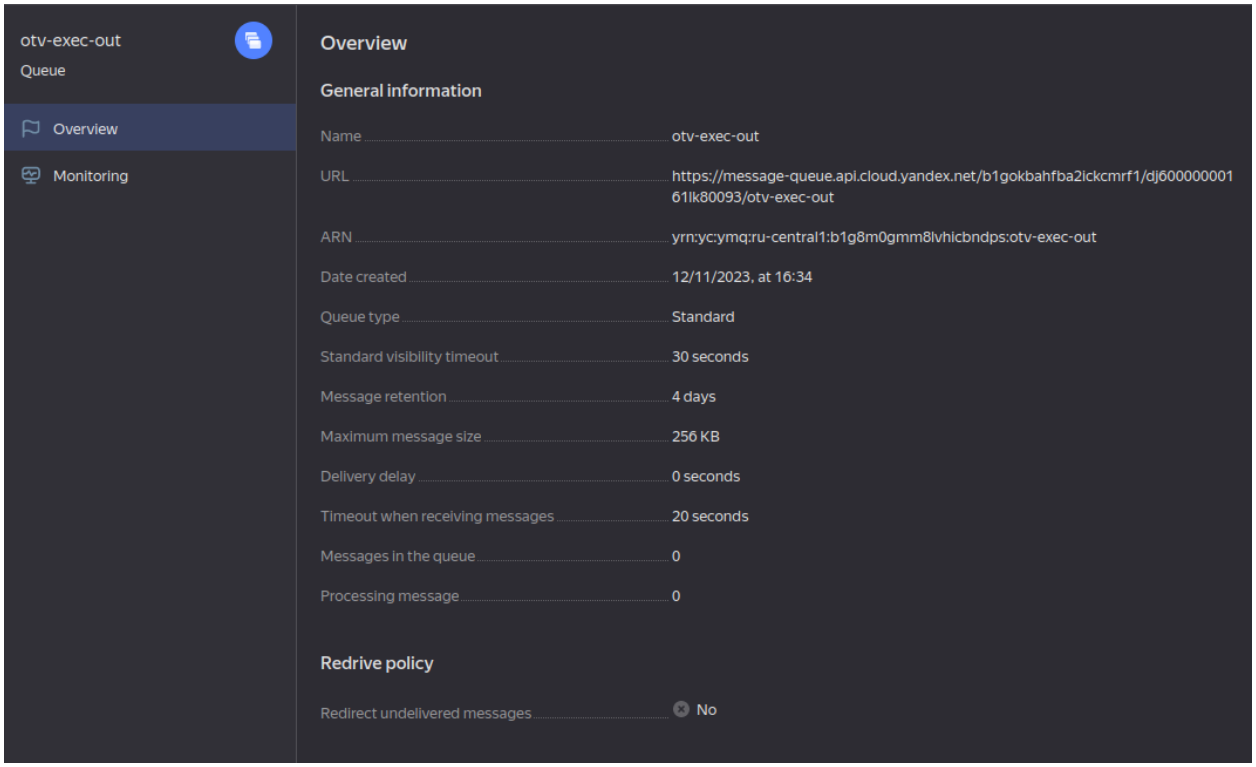
В системе используется две очереди:
“otv-exec-in” - очередь для отправки запросов на исполнение пайплайна.



Формат запросов:
{“resUri”: string, “executionId”: string}

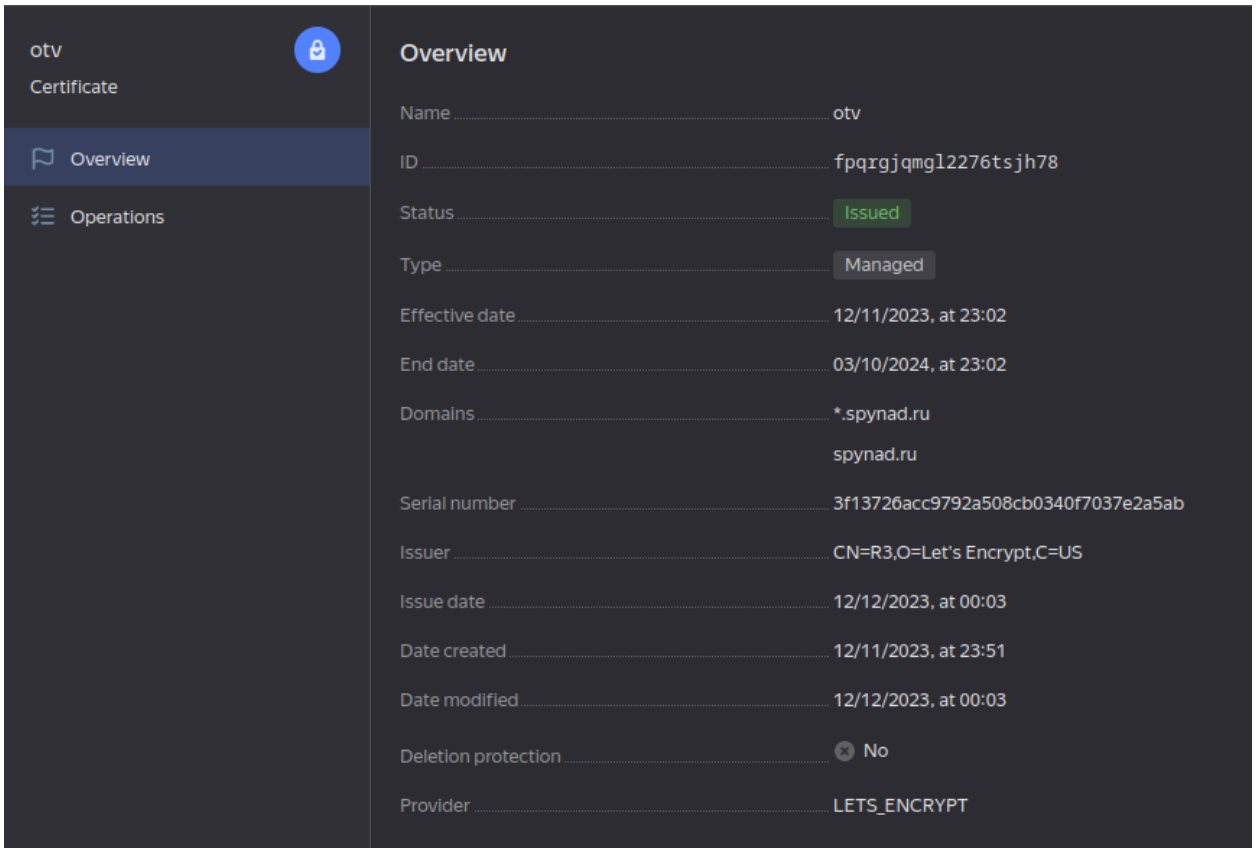
resUri - ссылка на объект S3 с пайплайном
executionId - уникальный идентификатор выполнения пайплайна (UUID)

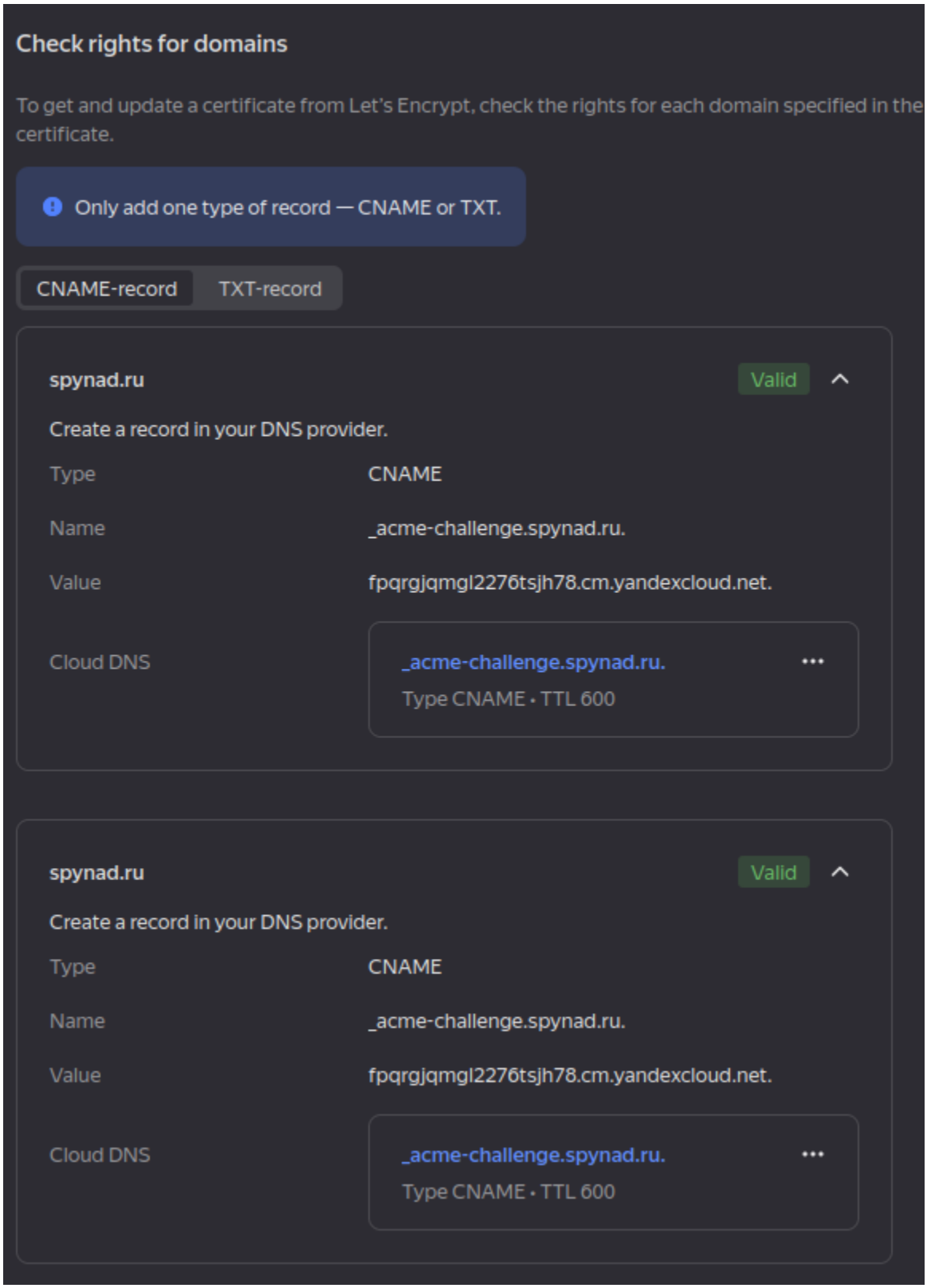
“otv-exec-out” - очередь для отправки “результатов” выполнения пайплайна.



Формат результатов:
{“executionId”: string, “executionUri”: string, “success”: boolean}
executionId - уникальный идентификатор выполнения пайплайна (UUID)
executionUri - ссылка на объект S3 с непосредственно результатами выполнения пайплайна (инференции)
success - поле, указывающее на успешность выполнения (если при выполнении произошла ошибка, то в первых двух полях будут значения null)

Certificate Manager





Регистратор домена - reg.ru, основные nameservers - Cloudflare.

Monitoring service

Для сбора метрик с виртуальной машины и Main Service используется Yandex Unified Agent, установленный на машину.

Dashboard метрик нашего сервиса:

Также мониторинг предоставляет готовые dashboards по облачным сервисам. Например:

Message Queue

Certificate Manager

Масштабирование

Узкое место в описанной системе - это виртуальная машина с GPU и backend-ом собственной API инференции. На одной машине может одновременно выполняться только одна задача - либо генерация изображений, либо генерация текста. Ситуацию может изменить:

- добавление дополнительных GPU на машины - в таком случае можно на каждый графический процессор назначить по одной задаче инференции нейронной сети.
- увеличение количества машин с GPU - каждая машина будет выбирать по задаче из очереди и обрабатывать запрос.

Если объединить эти два способа, можно добиться значительного прироста количества обрабатываемых задач за определенное время

При увеличении количества пользователей можно развернуть/настроить авто скейлинг машин с Frontend/Backend (задачи попадают только на одну случайную машину), перед ними поставить балансировщик нагрузки, например, на основе Network Load Balancer (Yandex Cloud)

Ссылки

Веб-приложение: https://spynad.ru/
Бекенд: https://backend.spynad.ru/swagger-ui/index.html

Эндпоинты swagger