Инструкция по локальному развороту платформы распознавания номеров

Эта инструкция применяется при локальном развертывании платформы распознавания номеров на сервере заказчика. При разворачивании устанавливаются только минимально необходимые для работы платформы микросервисы. Микросервисы пользовательского интерфейса (фронтенд), а также микросервисы платформы, не связанные с распознаванием номеров, не устанавливаются. Данный вариант установки, предполагает интеграцию платформы распознавания номеров с ПО заказчика. При этом платформа получает видеопоток с камер, обрабатывает его, детектирует транспортные средства и автомобильные номера и передаёт результаты детекаций на сервисы заказчика посредством вебхуков.

Помимо основных сервисов, также необходимо установить "инфраструктурные" сервисы, такие как базы данных, хранилища и брокеры сообщений, которые обеспечивают необходимую инфраструктуру для работы основных сервисов.

Устанавливаемые "инфраструктурные" сервисы:

1. **PostgreSQL:** Реляционная база данных.
2. **MongoDB:** Документо-ориентированная NoSQL база данных.
3. **Minio:** Объектное хранилище (S3-совместимое).
4. **Redis:** Кеширующее хранилище данных.
5. **RabbitMQ:** Брокер сообщений.
6. **Triton Server:** Сервер Triton, который является частью платформы NVIDIA Triton Inference Server, предназначен для развертывания и выполнения моделей машинного обучения и инференса.

Устанавливаемые микросервисы платформы (см. Приложение):

1. **nv-watcher-go:** Сервис получает видеопотоки с камер и нарезает их на кадры.
2. **nv-analitics-router:** Сервис маршрутизации кадров.
3. **nv-plates:** Сервис для работы с результатами распознавания автомобильных номеров.
4. **nv-cameras:** Сервис управления камерами.
5. **nv-platform-controller**: Сервис для работы со сценариями.
6. **nv-reactions**: Сервис для работы с реакциями.
7. **nv-security-api**: Сервис управления пользователями, ролями и правами доступа.
8. **nv-plates-bl-controller**: Сервис взаимодействия с моделями машинного обучения.

Платформа передаётся заказчику в виде архива с дистрибутивом, для локальной установке на сервере заказчика.

Пайплайн платформы распознавания номеров (см. Приложение):

Видеопоток, поступающий на сервис **nv-watcher-go**, разбивается на отдельные кадры. Эти кадры затем передаются на сервис **nv-analytics-router**, который направляет их на соответствующий сервис в соответствии с сценарием, предоставленным сервисом **nv-platform-controller**.

В данной локальной установке единственный сценарий предполагает направление всех кадров на сервис **nv-plates**. Здесь кадры помещаются в очередь с названием plate-bl_task брокера сообщений. Сервис **nv-plates-bl-controller** извлекает кадры из очереди и направляет их на модель машинного обучения для анализа.

После обработки моделью, сервис **nv-plates-bl-controller** передает результат в очередь plate-bl_result брокера сообщений. Далее этот результат забирается сервисом **nv-plates** и направляется обратно на сервис **nv-platform-controller** для дальнейшей обработки в соответствии с сценарием. На основе этой обработки отправляется запрос на запуск реакции на сервис **nv-reactions**.

Наконец, сервис реакции **nv-reactions** отправляет вебхук на сервис заказчика для уведомления о реакции.

Содержимое архива с дистрибутивом платформы:

9. **infra:** Конфигурационные файлы "инфраструктурных" сервисов.
10. **services:** Конфигурационные файлы микросервисов платформы.
11. **weight:** Веса моделей машинного обучения.
12. **src:** Исходные коды микросервисов платформы.

Последовательность локального развёртывания платформы распознавания номеров:

1. Запуск "инфраструктурных" сервисов.
2. Запуск конфигурационных скриптов, для создания необходимых записей в БД.
3. Загрузка в хранилище **Minio** прилагаемых весов моделей машинного обучения.
4. Перезапуск **Triton Server** (что бы **Triton** загрузил веса)
5. Конфигурирование брокера сообщений **RabbitMQ**.
6. Запуск микросервисов платформы.
7. Создание организацию и привязанного к ней юзера с необходимыми правами в сервисе **nv-security-api**.
8. Генерация jwt-токена для доступа к сервисам **nv-cameras**, **nv-platform-controller**, **nv-reactions**.
9. Добавление камер в сервис **nv-cameras**.
10. Добавление реакции в сервис **nv-reactions**.
11. Добавление сценария в сервис **nv-platform-controller**.
12. Перезапуск сервисов платформы распознования номеров.

Запуск "инфраструктурных" сервисов

Из каталога infra, сервисы запускаются командой:

```
docker compose up -d
```

Запуск конфигурационных скриптов, для создания необходимых записей в БД

Запускаем скрип, создающий юзеров и базы данных в Postgresql, необходимые для работы микросервисов **nv-security-api**, **nv_reactions**, **nv-cameras**, **nv-plates**, **nv-platform-controller**:

```
docker exec -it infra-postgresql-1 psql -U postgresql -d
postgresql -f /init.sql
```

Загрузка в хранилище Minio прилагаемых весов моделей машинного обучения

Открываем Web-интерфейс **Minio**

http://{IP сервера}:9100

Username: nv-storage

Password: nv-storage

Создаём бакет triton-weights. В этот бакет загружаем все веса из папки weight дистрибутива, сохраняя относительные пути.

Перезапуск Triton Server

Из каталога infra, перезапускаем **Triton Server** командой:

```
docker compose restart triton
```

Конфигурирование брокера сообщений RabbitMQ

Открываем Web-интерфейс брокера

http://{IP сервера}:15672

Username: dispatcher

Password: dispatcher

Переходим на вкладку Admin, в разделе Add a user добавляем юзера bl-model с паролем bl-model, переходим в раздел Virtual Hosts. Создаём новый vhost с именем models. Далее в таблице выбираем новый vhost и в разделе Permissions устанавливаем права на vhost для юзера bl-model.

Запуск микросервисов платформы

Из каталога services, сервисы запускаются командой:

```
docker compose up -d
```

Создание организацию и привязанного к ней юзера с необходимыми правами в сервисе nv-sevurity-api

Переходим на web-интерфейс сервиса **nv-security-api**:

http://{IP сервера}:8081/docs

В разделе Organization, выбираем Add Organization и добавляем новую организацию. Записываем id созданной организации.

В разделе Roles, добавляем роль cameras, устанавливаем ей следующий список permissions

["cameras.camera.get", "cameras.camera.put", "cameras.camera.post","cameras.area.put", "cameras.camera.delete", "cameras.area.delete", "cameras.area.get", "cameras.area.post"]

Записываем id созданной роли.

Аналогично, создаём роль plates со следующими permissions

["plates.plate.post", "plates.detection.get", "plates.detection.delete", "plates.plate.get", "plates.catalog.delete", "plates.plate.delete", "plates.catalog.put", "plates.plate.put", "plates.catalog.post", "plates.catalog.get"]

scenarios со следующими permissions

["scenarios.area.delete", "scenarios.scenario.delete", "scenarios.area.post", "scenarios.scenario.put", "scenarios.area.get", "scenarios.area.put", "scenarios.scenario.post", "scenarios.scenario.get"]

reactions со следующими permissions

["reactions.reaction.get", "reactions.reaction.post", "reactions.reaction.delete", "reactions.reaction.put", "reactions.history.get"]

В разделе Users, выбираем Add User и добавляем нового юзера, указываем id созданной организации и список id созданных ролей.

Генерация jwt-токена для доступа к сервисам nv-cameras, nv-platform-controller, nv-reactions

Переходим на web-интерфейс сервиса **nv-security-api**:

http://{IP сервера}:8081/docs

В разделе Authorization, выбираем Login For Access Token и авторизируемся используя email и пароль созданного юзера. В ответе будет jwt-токен, который необходим для доступа к сервисам **nv-cameras**, **nv-platform-controller**, **nv-reactions**.

Добавление камер в сервис nv-cameras
Переходим на web-интерфейс сервиса nv-cameras:

http://{IP сервера}:8083/docs

В Authorize, указываем сгенерированный jwt-токен.

В разделе Cameras, выбираем Create Camera и добавляем камеру, указав в поле connection адрес rtsp потока, в поле fps, указываем количество кадров в секунду, в поле resolution — разрешение камеры.

Аналогичным образом добавляем остальные камеры.

Добавление реакции в сервис nv-reactions
Переходим на web-интерфейс сервиса **nv-reactions**:

http://{IP сервера}:8082/docs

В Authorize, указываем сгенерированный jwt-токен.

В разделе Reactions выбираем Create Reaction. В поле type указываем тип webhook, в поле config указываются настройки вебхука, такие как HTTP метод, url, поля заголовка запроса, параметры запроса, данные передаваемые в запросе (например детектированный номер ТС).

Пример правильной настройки вебхука:

```
"method": "post",
"url": "https://our-customer.ru/api/v1/open_gate",
"headers": {
  "Authorization": "Basic gmV2cm9jb3JlOl5ldXJvY29yZQ=="
},
"params": null,
"body": {
  "format": "json",
  "data": [
    {
      "type": "int",
      "key": "building_id",
      "value": "2039"
    },
    {
      "type": "str",
      "key": "plate",
      "value": "{{plate}}"
    }
  ]
}
```

Добавление сценария в сервис nv-platform-controller
Переходим на web-интерфейс сервиса **nv-platform-controller**:

http://{IP сервера}:8084/docs

В Authorize, указываем сгенерированный jwt-токен.

В разделе Scenarios выбираем Create Scenario указываем id камеры, «зоны интересов» (области в которой будут происходить детекации), id реакции и другие опции сценария (список опций указан в подсказке на страницы создания сценария).

Перезапуск сервисов платформы распознования номеров

Проверяем, все ли микросервисы платформы запущены. Если некоторые сервисы оказались не запущены, запускаем их. Такое может случиться, например с сервисом **nv-watcher-go**, потому что на момент его запуска ещё не были заведены камеры.

```
docker compose up -d nv-watcher-go
```