README.md

SLR AI-Agent

Универсальный агент шины обработки ИНС.

HTTP API

Установка и запуск

Docker image

1. Загрузить последний актуальный образ:
   `docker pull registry.gitlab.com/softlogicrus/ai-agent:latest`
2. Запустить командой:
   `docker start -d --name ai-agent -p 8080:80 -v /etc/ai-agent/your_config.yml:/etc/ai-agent/config.yml:ro registry.gitlab.com/softlogicrus/ai-agent:latest`

Установка из deb-пакета

1. Загрузить deb-пакет требуемой версии.
2. Установить ПО командой:
   `sudo dpkg -i ai-agent_<VERSION>_<ARCH>.deb`
3. Поместить файл `config.yml` в директорию `/etc/ai-agent` .
4. Перезапустить сервис командой `sudo systemctl restart ai-agent` .

Сборка из исходного кода

Если на ЦУ нет установленного OpenCV > 4.0.0, выполните команду `make preinstall && make build_opencv` . При сборке для вычислений на GPU установите параметры GPU=1 и CUDA_ARCH_BIN. CUDA_ARCH_BIN зависит от модели ГП. Узнать можно на официальном сайте NVIDIA.

1. Перейдите в корневую директорию проекта и выполните:
   `make build  sudo make install`
2. Создайте файл `config.yml` в директории `/etc/ai-agent` .
3. Сконфигурируйте систему под Ваши нужды и выполните
   `sudo systemctl restart ai-agent` .

Конфигурирование

В агенте существует несколько базовых элементов:

1. Входные точки (**Entrypoints**)
2. Вычислители (**Estimators**) и Ядра (**Cores**)
3. Выходные точки (**Endpoints**)
4. Схемы (**Schemas**)
5. Хранилища (**Storages**) - хранилища Redis, используемые системой. Здесь инструкция по установке. Также доступен официальный Docker-образ.

Существует основная обязательная секция **service**. Пример заполнения:

```yaml
service:
  sync_period: 5 # Период синхронизации с внешними источниками видео (в секундах)
  core_response_timeout: 60 # Максимальная задержка получения ответа от ядра ИНС (в секундах)
  image_chunk_size: 1048576 # Размер фрагмента изображения, передаваемого по gRPC ядрам ИНС (в байтах)
  listener:
    # HTTP API сервис
    http:
      enabled: true
      host: localhost
      port: 8000
    # gRPC API сервис
    grpc:
      enabled: false
      host: localhost
      port: 20100
      settings:
        use_ssl: false
        ssl_cert_path: "keys/certificate.pem"
        ssl_key_path: "keys/server.key"
  # Настройки логирования
  logging:
    level: "info"
    formatter: "text" # text или json
    handler: "file" # console или file
    path: "/var/log/ai-agent" # путь к директории логов
```

Entrypoints

Какой-либо источник изображений или видеоданных.
Пример конфигурирования:

```yaml
entrypoints:
  example:
    uri: "/etc/ai-agent/entrypoints" # путь к файлу или URL
    type: "file" # file или http
    # список используемых схем
    sources:
      - src
    # список имен выходных точек
    endpoints:
      - file
      - http
```

Estimators & Cores

Совокупность алгоритмических ядер (**cores**), работающих по единому gRPC API. Ядро - 1 алгоритмическая единица. Имеет адрес и порт, а также тип. Поддерживаются следующие типы:

- *NW_KIND_DETECTION* - детектор
- *NW_KIND_RECOGNITION* - распознавание
  Пример конфигурирования:

```yaml
estimators:
  13: # Название вычислителя
    cores: # Список ядер
      - kind: AI_AGENT_EVENT_TYPE_DETECTION
        host: localhost
```

```
        port: 33338
    settings:
      use_storage: false # Включить/отключить накопление через хранилище. Работает только для анализа видеопотока
      storage_name: main # Имя хранилища
      accumulate_each: "1m" # Временной интервал накопления
```

На данном этапе данные после накопления фильтруются только по параметру максимального показателя уверенности.

Endpoints

Сервисы, которые будут принимать результаты обработки.
Поддерживаются следующие типы:

- **fs** - запись в файловую систему. При этом результаты записываются в файл `report-YYMMDD.csv`, а изображения в директорию `images/YYMMDD`.
- **http** - отправка по протоколу HTTP/HTTPS.
  Пример конфигурирования:

```
endpoints:
  file:
    type: fs # запись результатов в файловую систему
    uri: /etc/ai-agent/media # путь к директории
    with_dri: true # отображать/скрыть поле "dri" в ответе
    with_properties: true # отображать/скрыть поле "properties" в ответе
  http:
    type: "http" # отправка во внешнюю систему
    uri: https://example.com
    sinks:
      - example_sink # schema
    with_dri: false
    with_properties: true
    client_settings: # настройки http-клиента
      timeout: "5s" # максимальная задержка при отправке
      headers: # HTTP-заголовки
        Content-Type: "application/json"
```

Schemas

Описание API источника или приемника. Содержат тип (на данный момент только JSON), метод (GET или POST). Поле *required_data* содержит информацию о полях, которые нужно извлечь из данных, полученных от источника. Вложенность обозначается точкой: `.`, элемент списка - знаком `[*]`.
Например, в схеме

```
{"items": [
      {
        "id": "foo",
        "name": "bar"
      }
  ]
}
```

для извлечения параметра `id` нужно использовать шаблон
`items.[*].id`.
Пример конфигурирования:

```
schemas:
  example_sink:
```

```
    type: json
    method: post # HTTP-метод
    url: "/events" # URL. Будет соединен с параметром "uri" в endpoints

  example_source:
    type: "json"
    method: "get"
    url: "test.json"
    required_data:
      - type: "camera" # на данный момент только тип "Камера"
        fields:
          - key: "id" # идентификатор, может быть строкой или числом
            pattern: "[*].id" # шаблон для поиска
          - key: "url"
            pattern: "[*].options.url" # ссылка на видеопоток
          - key: "estimators"
            pattern: "[*].options.estimators.[*]" # список вычислителей
          - key: "frequency" # частота анализа (в секундах)
            pattern: "[*].options.freq"
```