EasyHomeCloud HTTP API v0.7

Оглавление

1. Общая информация

Исходная задача состоит в том, чтобы система EasyHomeCloud передавала информацию о регистрах контроллеров клиентам и от клиентов – контроллерам, создавая связь многие ко многим. Более того, EasyHomeCloud создает возможность подключения клиентов к контроллерам, у которых динамически изменяемый адрес – необходимо подключаться к известному серверу EasyHomeCloud.

Изначально клиент реализован на стеке C++/Qt устанавливаемым приложением EasyHome на Windows, Linux, Android, Mac OS. Общение напрямую реализовано по протоколу Modbus TCP. Поэтому, EasyHomeCloud, в первую очередь, реализует трансляцию таких запросов. Это позволяет выделить статический IP-адрес только для сервера с EasyHomeCloud, к которому подключаются контроллеры и клиенты, имеющие любой IP-адрес (без EasyHomeCloud контроллеры, к которым подключаются клиенты напрямую, должны иметь статический IP-адрес).

Следующий шаг – это адаптация EasyHomeCloud под работу с web-клиентами. Для этого было создано настоящее HTTP API для клиента. Система EasyHomeCloud имеет возможность управления контроллерами посредством HTTP 1.1 POST и GET запросов с телом в формате JSON.

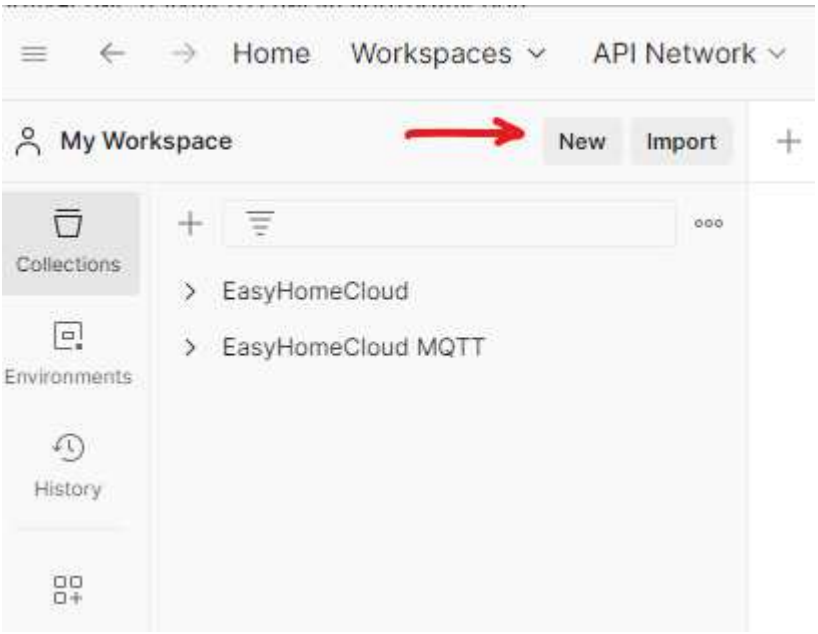
Поскольку в интерфейсах клиентов существуют элементы, которые необходимо ежесекундно обновлять, возникает потребность введения протокола MQTT, который снимет с клиентов и сервера нагрузку, возникающую при ежесекундном запросе от клиента по каждому необходимому регистру (особенно в случае, если несколько клиентов спрашивают одни и те же регистры или их диапазоны). Вместо этого, клиент выполняет подписку на регистры, которые затем читает их из MQTT брокера, в который публикует ПО EasyHomeCloud по мере изменения данных регистров. Поскольку существует проблема перегрузки сервера EasyHomeCloud запросами на уже устаревшие ненужные и неактуальные регистры, существует механизм очистки таких регистров по истечении времени подписки или отключении всех клиентов от ПЛК.

Сама подписка реализуется клиентом с помощью HTTP запроса с соответствующим параметром.

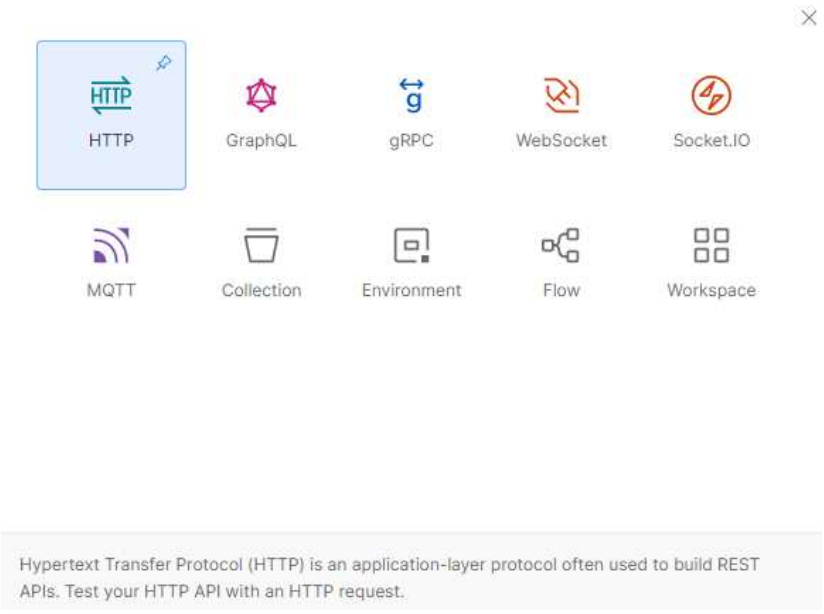
2. Интерфейс для тестов HTTP и MQTT Postman

Установка и настройка (HTTP)

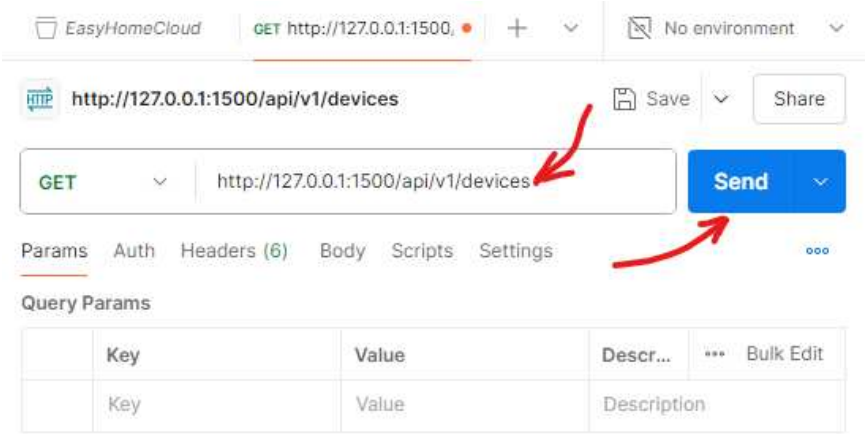
1. Скачать с официального сайта Postman и установить
2. Создать новый запрос



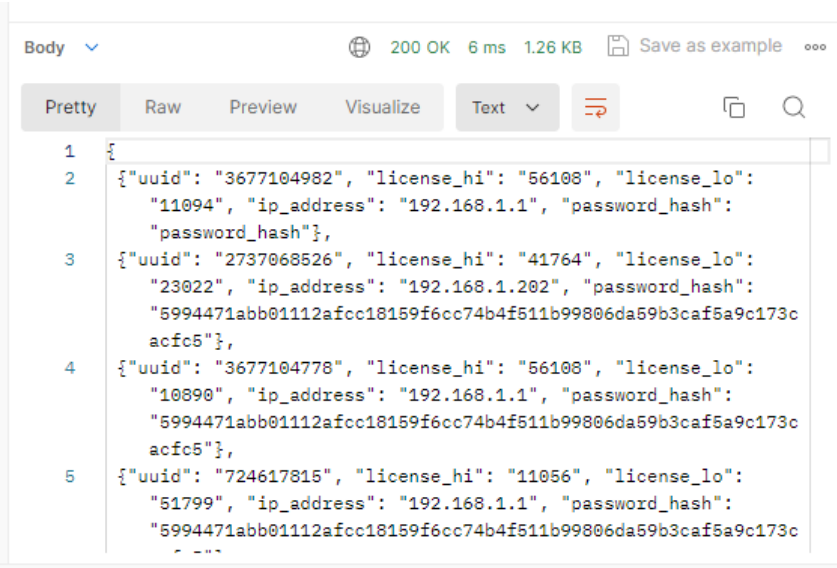
3. Выбрать HTTP запросы



4. Заполнить поле сервера, например **http://127.0.0.1:1500/api/v1/devices** и нажать **Send**



Это позволит отправить GET запрос и получить ответ от сервера в виде списка известных ему устройств в поле вывода:



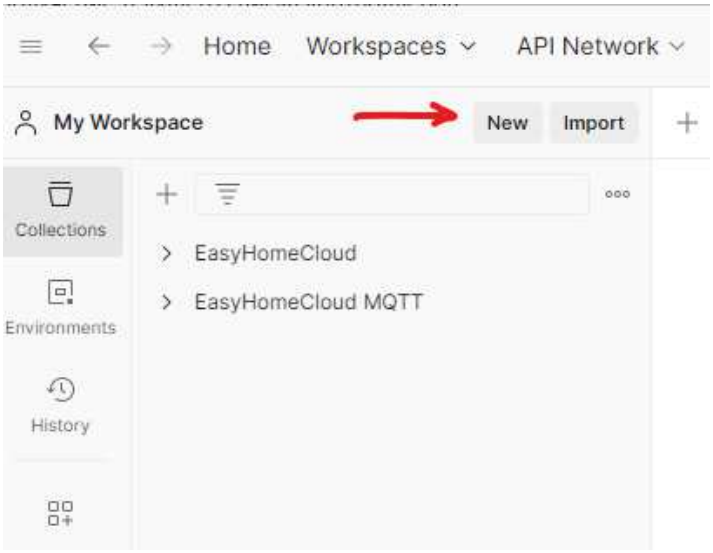
Пример HTTP-запроса, сформированного и отправленного через программу Postman:

```
POST /api/v1/device_request HTTP/1.1
Host: 127.0.0.1:1500
Content-Type: application/json
Content-Length: 185
```

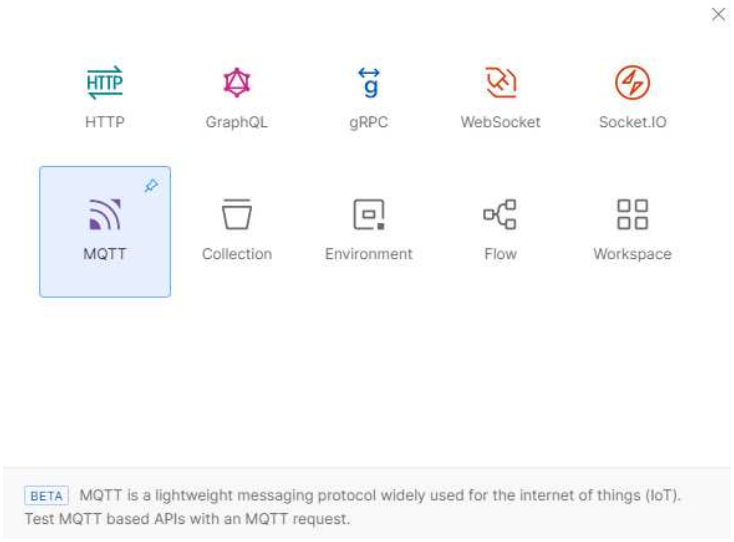
```
{
    "password": "5994471abb01112afcc18159f6cc74b4f511b99806da59b3caf5a9c173cacfc5",
    "request_type": "get",
    "licenseID_hi": 17111,
    "licenseID_lo": 20836,
    "word": {
        "range_begin": 310,
        "range_end": 314,
        "list": [315, 316]
    }
}
```

Настройка (MQTT)

1. Создать новый запрос



2. Выбрать MQTT запросы



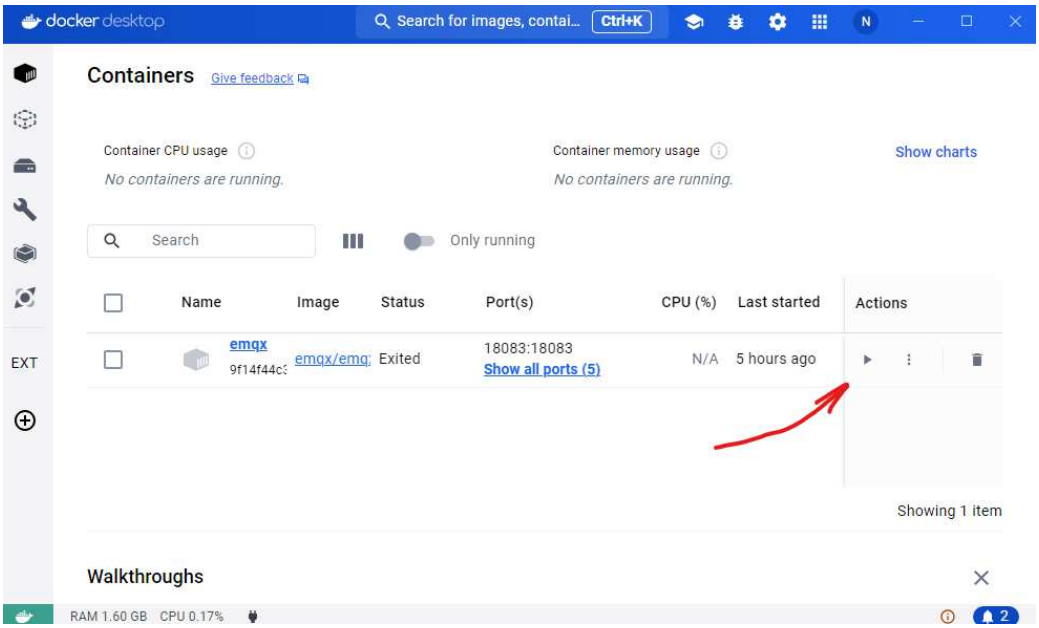
3. Заполнить поле брокера, например **mqtt://127.0.0.1:1883**

4. Заполнить данные пользователя во вкладке **Authorization** и нажать Connect

Подключение выполнится успешно, если сервер MQTT брокера запущен.

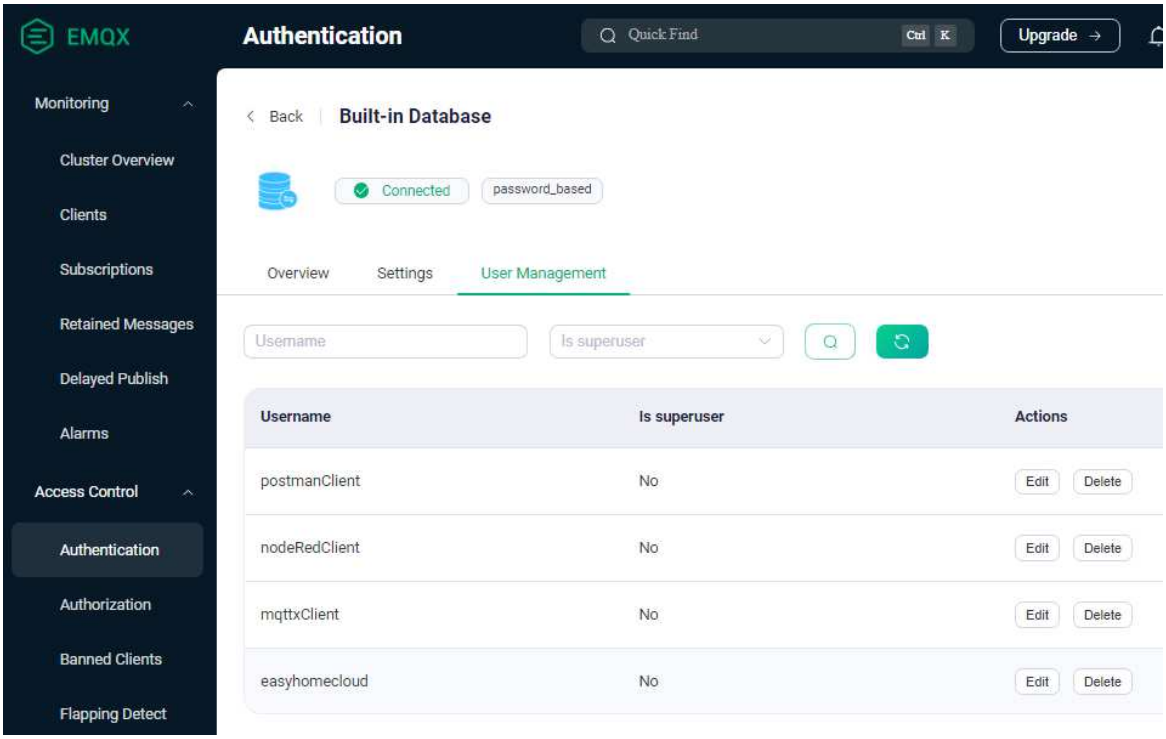
Запуск MQTT брокера (EMQX)

1. Установить Docker Desktop для развертывания MQTT брокера
2. Установить Docker контейнер MQTT брокер EMQX
3. Запустить брокер



4. Подключиться через браузер по адресу http://127.0.0.1:18083/
5. Выполнить регистрацию/авторизацию

6. Добавить клиентов (в том числе для тестов через Postman)
    a. Для этого перейти во вкладку **Access Control -> Authentication**
    b. В правом верхнем углу нажать **Create**. Выбрать **Mechanism – Password-Based**, **Backend – Built-in Database**, **Configuration** оставить заполненной по-умолчанию. Нажать **Create.**
    c. В поле **Action** нажать на **Users** и заполнить поля, добавив необходимых клиентов.



Например,

Для клиента EasyHomeCloud имя пользователя и пароль выбраны – easyhomecloud (указывается в settings.ini),

Для клиента Node-red – nodeRedClient – указывается для блоков **mqtt** в Node-red во вкладке **Безопасность**.

Запуск MQTT брокера (Mosquitto)

1. Установить брокер Mosquitto
2. Запустить брокер (для Windows – обычно C:\Program Files\mosquitto\mosquitto.exe)

3. Получение текущего списка контроллеров

Запрос на получение информации о списке контроллеров

GET запрос:

```
GET /api/v1/devices HTTP/1.1
Host: 127.0.0.1:1500
```

Пример ответа:

```
HTTP/1.1 200 OK
{
    {
        "uuid": "3677104778",
```

```
        "license_hi": "56108",
        "license_lo": "10890",
        "ip_address": "192.168.1.1",
        "password_hash": "5994471abb01112afcc18159f6cc74b4f511b99806da59b3caf5a9c173cacfc5"
    },
    {
        "uuid": "1121407332",
        "license_hi": "17111",
        "license_lo": "20836",
        "ip_address": "192.168.1.204",
        "password_hash": "5994471abb01112afcc18159f6cc74b4f511b99806da59b3caf5a9c173cacfc5"
    }
}
```

Получение текущего состояния контроллера

Шаблоны запроса на получение информации

Через POST запрос:

Получить **диапазон** элементов. Тип индекса или **byte**, или **word**:

```
POST /api/v1/device_request
{
    "password": "<sha256(пароль)>",
    "request_type": "get",
    "licenseID_hi": <верхний байт идентификатора>,
    "licenseID_lo": <нижний байт идентификатора>,
    "<тип индекса>": {
        "range_begin": <начальное значение>,
        "range_end": <конечное значение>
    }
}
```

Получить **список** элементов. Тип индекса или **byte**, или **word**:

```
POST /api/v1/device_request
{
    "password": "<sha256(пароль)>",
    "request_type": "get",
    "licenseID_hi": <верхний байт идентификатора>,
    "licenseID_lo": <нижний байт идентификатора>,
    "<тип индекса>": {
        "list": [<значение_n>, ..., <значение_k>]
    }
}
```

Получить **диапазон** и **список** элементов. Тип индекса или **byte**, или **word**:

```
POST /api/v1/device_request
{
    "password": "<sha256(пароль)>",
    "request_type": "get",
    "licenseID_hi": <верхний байт идентификатора>,
```

```
    "licenseID_lo": <нижний байт идентификатора>,
    "<тип индекса>": {
        "range_begin": <начальное значение>,
        "range_end": <конечное значение>,
        "list": [<значение_n>, ..., <значение_k>]
    }
}
```

Пример:

```
POST /api/v1/device_request HTTP/1.1
Host: 127.0.0.1:1500
Content-Type: application/json
Content-Length: 307
```

```
{
    "password": "5994471abb01112afcc18159f6cc74b4f511b99806da59b3caf5a9c173cacfc5",
    "request_type": "get",
    "licenseID_hi": 17111,
    "licenseID_lo": 20836,
    "word": {
        "range_begin": 310,
        "range_end": 312,
        "list": [315, 316]
    }
}
```

Ответ на запрос выглядит следующим образом:

В случае неверного запроса:

```
HTTP/1.1 400 BAD REQUEST
{ }
```

В случае успеха:

```
HTTP/1.1 200 OK
```

```
{
   <тип возвращаемого элемента>: [
        {
            "index": <индекс элемента>,
            "value": <значение элемента>
        },
        {
            "index": <индекс элемента>,
            "value": <значение элемента>
        }
    ]
}
```

Для типа **word** возвращаются значения регистров в регистровой индексации, для типа **byte** – значения байт в байтовой индексации.

Пример:

```
HTTP/1.1 200 OK
```

```
{
    "word": [
        {
            "index": 310,
            "value": 256
        },
        {
            "index": 311,
            "value": 0
        },
        {
            "index": 312,
            "value": 256
        },
        {
            "index": 321,
            "value": 519
        },
        {
            "index": 322,
            "value": 0
        }
    ]
}
```

4. Подписка на текущее состояние контроллера

Шаблоны запроса на подписку

Через POST запрос:

Подписаться на **диапазон** элементов. Тип индекса **word**:

```
POST /api/v1/device_request
```

```
{
    "password": "<sha256(пароль)>",
    "request_type": "subscribe",
    "licenseID_hi": <верхний байт идентификатора>,
    "licenseID_lo": <нижний байт идентификатора>,
    "word": {
        "range_begin": <начальное значение>,
        "range_end": <конечное значение>
    }
}
```

Подписаться на **список** элементов. Тип индекса **word**:

```
POST /api/v1/device_request
```

```
{
    "password": "<sha256(пароль)>",
    "request_type": "subscribe",
    "licenseID_hi": <верхний байт идентификатора>,
```

```
    "licenseID_lo": <нижний байт идентификатора>,
    "word": {
        "list": [<значение_n>, ..., <значение_k>]
    }
}
```

Подписаться на **диапазон** и **список** элементов. Тип индекса **word**:

```
POST /api/v1/device_request
```

```
{
    "password": "<sha256(пароль)>",
    "request_type": "subscribe",
    "licenseID_hi": <верхний байт идентификатора>,
    "licenseID_lo": <нижний байт идентификатора>,
    "word": {
        "range_begin": <начальное значение>,
        "range_end": <конечное значение>,
        "list": [<значение_n>, ..., <значение_k>]
    }
}
```

Пример:

```
POST /api/v1/device_request HTTP/1.1
Host: 127.0.0.1:1500
Content-Type: application/json
Content-Length: 307
```

```
{
    "password": "5994471abb01112afcc18159f6cc74b4f511b99806da59b3caf5a9c173cacfc5",
    "request_type": "subscribe",
    "licenseID_hi": 17111,
    "licenseID_lo": 20836,
    "word": {
        "range_begin": 310,
        "range_end": 312,
        "list": [315, 316]
    }
}
```

Ответ на запрос выглядит следующим образом:

В случае неверного запроса:

```
HTTP/1.1 400 BAD REQUEST
{ }
```

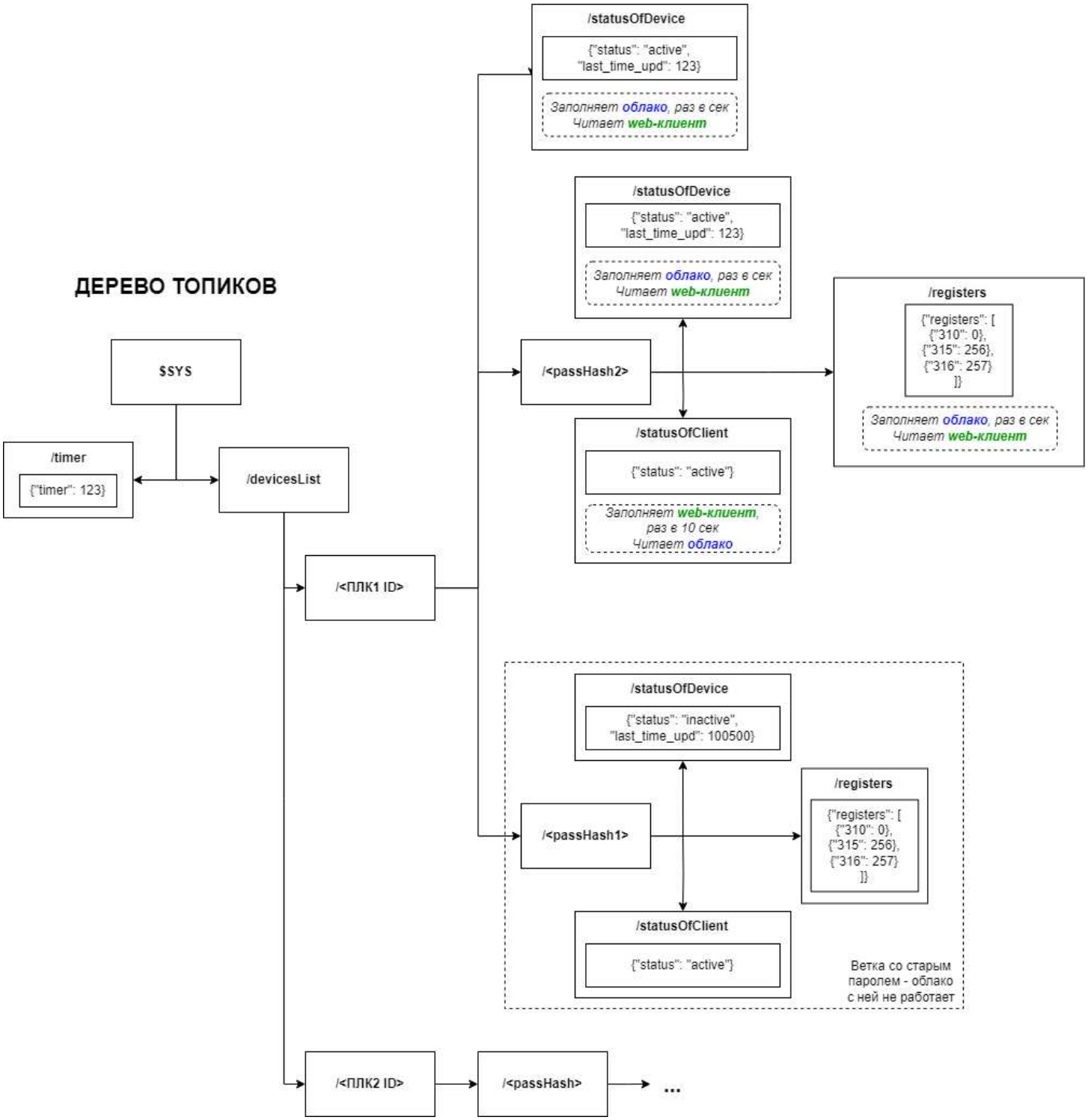
В случае успеха сервер запоминает список запрошенных регистров для мониторинга в ПЛК и публикации в MQTT брокере. Сервер сообщает адрес брокера, который указан в **settings.ini** в **MQTT_BROKER_URI**. Пример ответа:

```
HTTP/1.1 200 OK
```

```
{
   "result": "subscribed","mqttUri": "127.0.0.1:1883"
```

```
}
```

После отправки успешного запроса на подписку клиент должен подписаться на сам MQTT брокер (необходимые топики). Топики в брокере располагаются следующим образом:



- **$SYS** – Корневой топик
  - **/timer** – топик таймера с сообщениями JSON формата, содержащими ключ **timer** и целочисленным значением периода с момента запуска сервера EasyHomeCloud в миллисекундах. Пример: {“timer”: 123}
  - **/devicesList** – топик, содержащий топики устройств
    - **/<ПЛК ID>** – топик устройства, например /3677104982
      - **/statusOfDevice** – топик состояния устройства с сообщениями формата JSON, содержащими ключ **status** – строковое значение (**active** или

inactive) и ключ **last_time_upd** – целочисленное значение с отметкой времени с момента запуска сервера EasyHomeCloud в мс. Пример: {“status”: “active”, “last_time_upd”: 123}

- **/<passHash>** – топик-пароль, создаваемый в соответствии с хэшем пароля устройства, внутри которого размещается полезная информация об устройстве. С топиком старого пароля сервер EasyHomeCloud не взаимодействует. Например: /5994471abb01112afcc18159f6cc74b4f511b99806da59b3caf5a9c173cacfc5
    - **/statusOfDevice** – топик состояния устройства с сообщениями формата JSON, содержащими ключ **status** – строковое значение (**active** или **inactive**) и ключ **last_time_upd** – целочисленное значение с отметкой времени с момента запуска сервера EasyHomeCloud в мс. Пример: {“status”: “active”, “last_time_upd”: 123}. *ОТЛИЧИЕ от аналогичного топика выше в том, что при смене пароля на устройстве клиент не видит здесь обновлений, которые могут быть в топике выше и делает вывод о том, что пароль устройства сменился*
    - **/statusOfClient** – топик состояния клиентов с сообщениями формата JSON, содержащими ключ **status** – строковое значение (**active**). Чтобы сохранялась подписка клиентов на устройство (и чтобы сервер не очищал список регистров этого устройства), клиенты должны отправлять сообщение {“status”: “active”} в этот топик хотя бы раз за время **CLIENTS_LIFETIME** (переменная в settings.ini сервера, миллисекунды)
    - **/registers** – топик в который сервер отправляет сообщение с регистрами и их значениями, если их значения изменились за время **DATA_UPD_PERIOD** (settings.ini в мс). Например: {“registers”: [{“310”: 0}, {“313”: 256}, {“320”: 257}] }. (Адрес задается в регистровой адресации (word).

5. Изменение текущего состояния элементов системы

Шаблон запроса на изменение

Через POST запрос:

Для word или byte:

```
POST /api/v1/device_request
```

```
{
    "password": "<sha256(пароль)>",
    "request_type": "set",
    "licenseID_hi": <верхний байт идентификатора>,
    "licenseID_lo": <нижний байт идентификатора>,
    "<тип значения word или byte>": [
      {
        "index": <индекс>,        //адрес в количестве word (%MW) или byte (%MB)
        "value": <значение>,      //от 0 до 65535 для word и от 0 до 255 для byte
        "mask": <значение>        //опционально для сохранения исходными битов,
                                  //для которых значение маски равно 0,
                                  //от 0 до 65535 для word и от 0 до 255 для byte
      }
    ]
}
```

Для bit:

```
POST /api/v1/device_request
```

```
{
    "password": "<sha256(пароль)>",
    "request_type": "set",
    "licenseID_hi": <верхний байт идентификатора>,
    "licenseID_lo": <нижний байт идентификатора>,
    "bit": [
      {
        "index": <индекс байта>,        //адрес байта (%MB)
        "value": <значение>,            //от 0 до 1
        "bitIndex": <индекс бита>       //от 0 до 7
      }
    ]
}
```

Ответ на запрос выглядит следующим образом:

В случае неверного запроса:

```
HTTP/1.1 400 BAD REQUEST
{}
```

В случае успеха:

```
HTTP/1.1 200 OK
```

```
{
    "result": "set",
```

```
    "set_type": "<тип изменяемых элементов>" //word, byte или bit
}
```

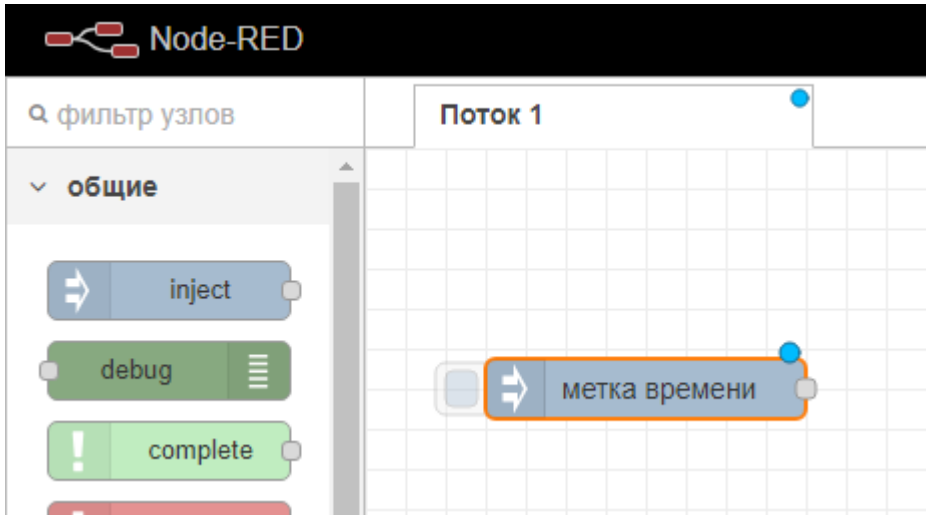
В случае неудачи:

```
HTTP/1.1 200 OK
{
    "result": "not set",
    "set_type": "<тип изменяемых элементов>" //word, byte или bit
}
```

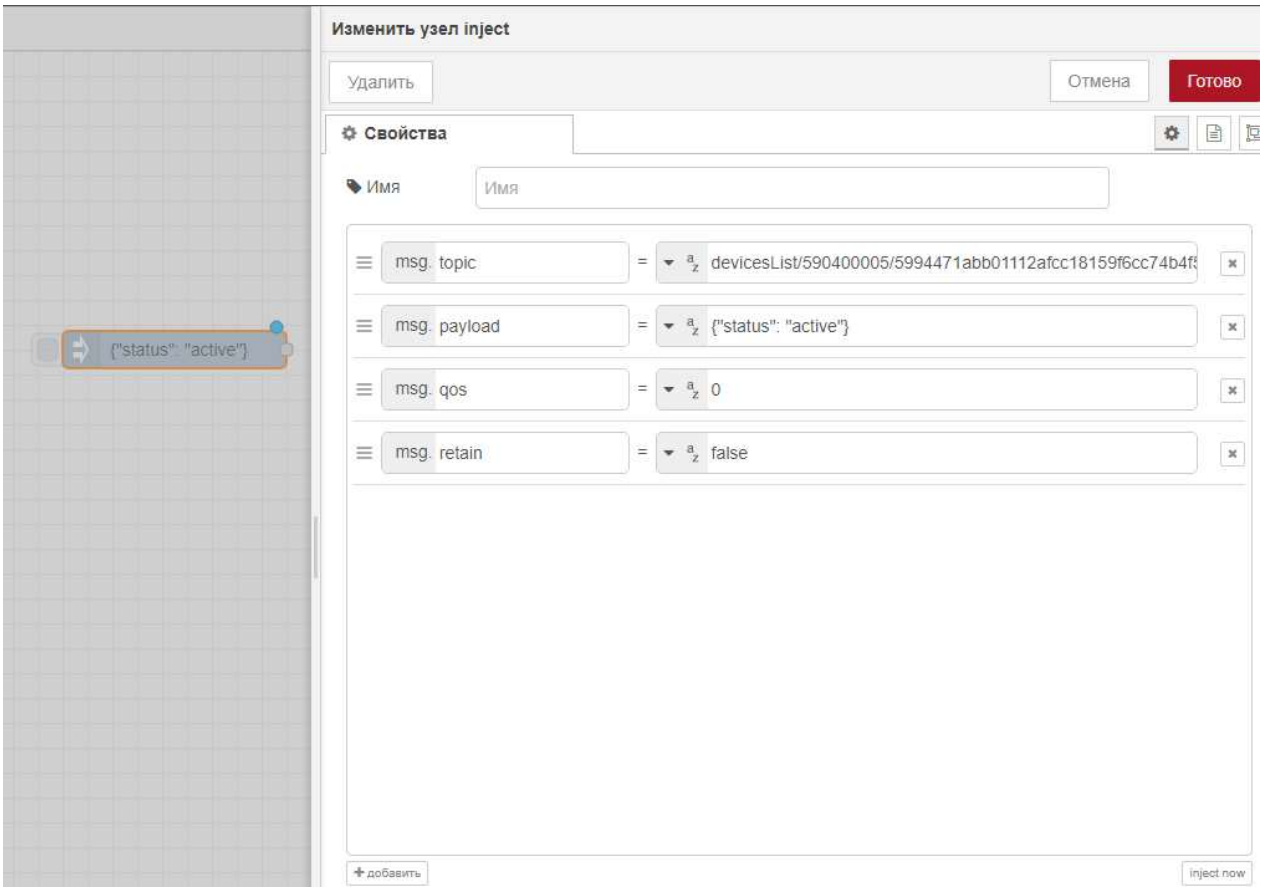
6. Тесты с помощью Node-red

Установка и настройка Node-red (Windows)

1. Выполните инструкции с официального сайта Node-red для установки
2. Запустите Node-red с помощью команды **node-red** в командной строке
3. Войдите в среду разработки через браузер **http://127.0.0.1:1880/**
4. Построим конструкцию, которая будет сообщать облаку о том, что хотя бы один клиент читает регистры устройства
   4.1. Из категории **общие** добавьте на поле блок **inject**, который будет генерировать объект с сообщением по нажатию на кнопку:

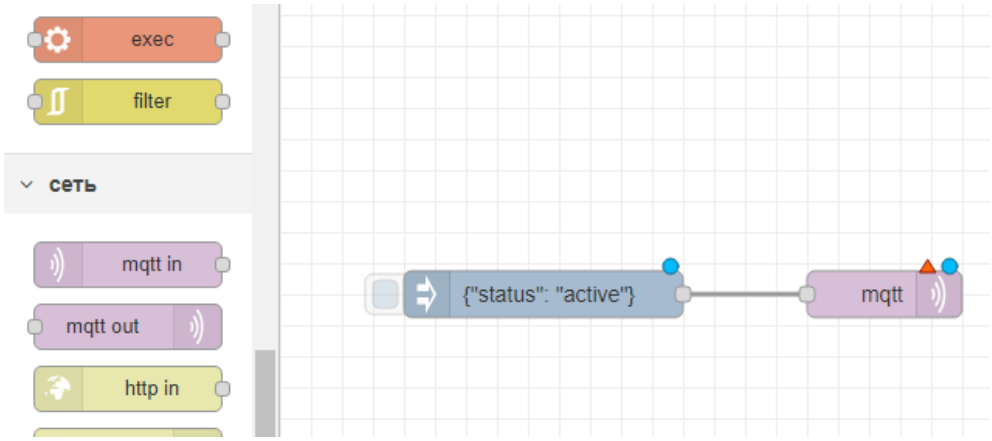


   4.2. Двойным нажатием ЛКМ откройте настройку блока и добавьте поля:
   a. topic = devicesList/*<ПЛК_ID>*/*<passwordHash>*/statusOfClient
   b. payload = {"status": "active"}
   c. qos = 0
   d. retain = false

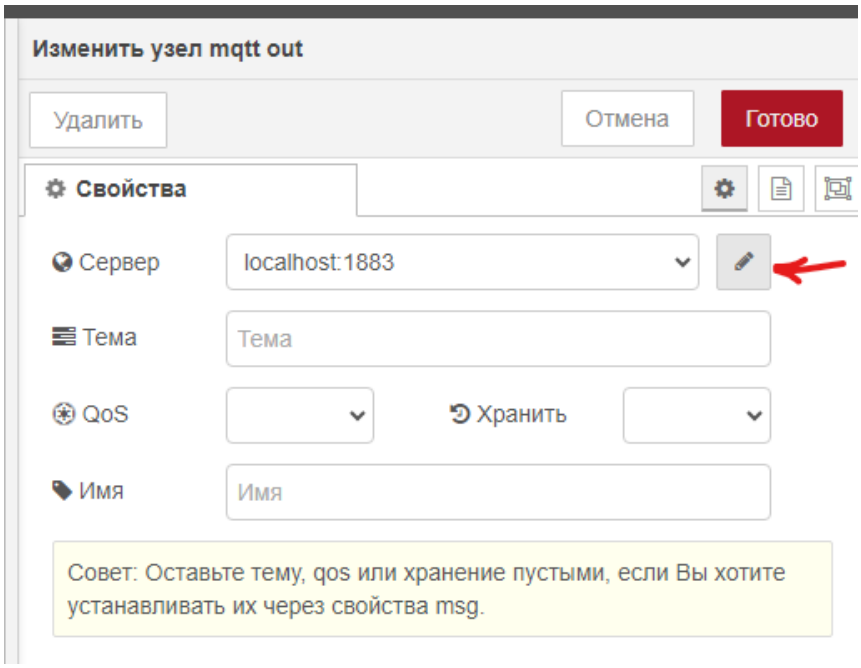


4.3. Нажать **Готово**

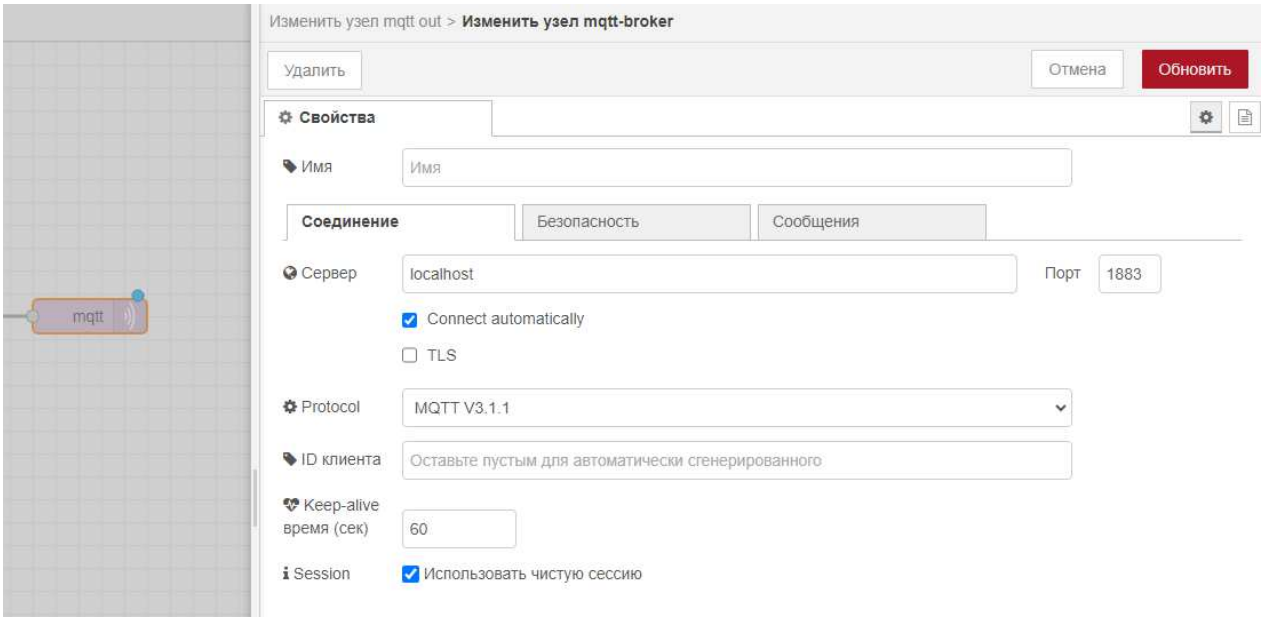
4.4. Добавить блок **mqtt out** из вкладки **сеть** и соединить его с блоком **inject**



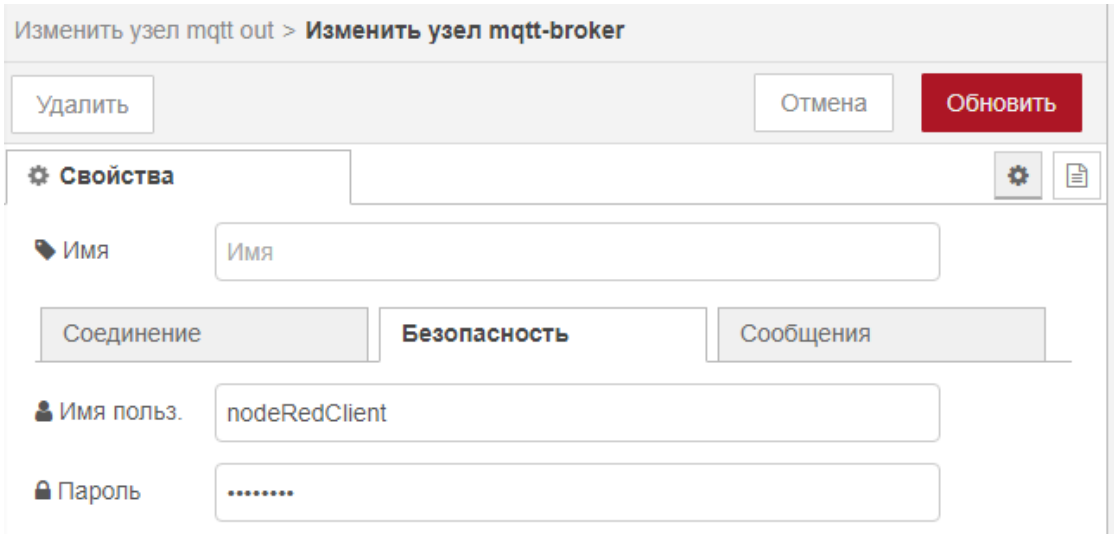
4.5. В настройках блока добавить MQTT сервер, нажатием на редактирование



4.6. В настройках сервера установить адрес сервера и порт, например **localhost** и **1883**

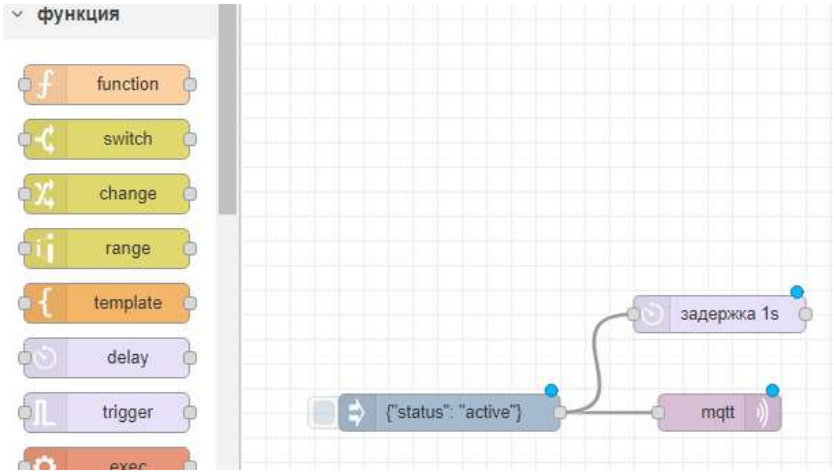


4.7. Во вкладке **Безопасность** укажем имя пользователя и пароль, как зарезервировали в настройках брокера (nodeRedClient)



5. Создадим модуль для отправки HTTP запроса для подписки на регистры

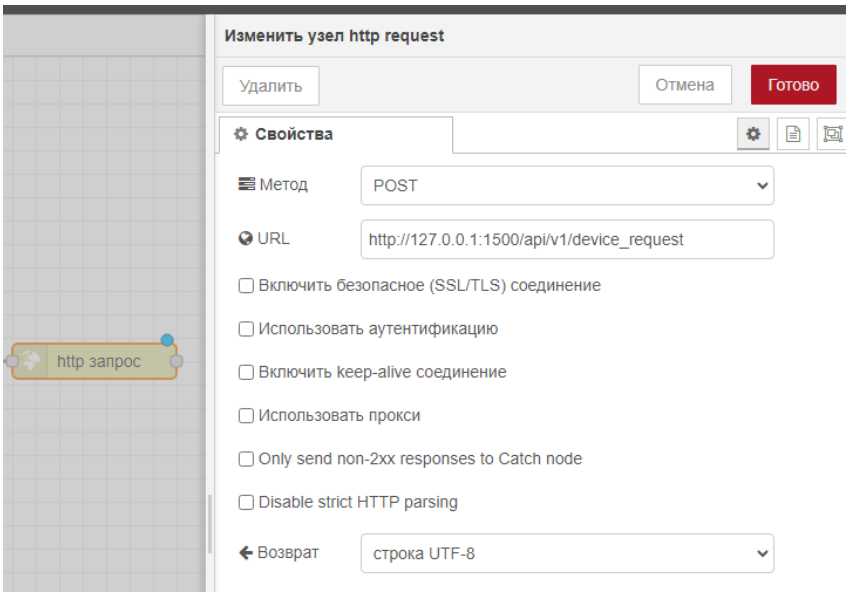
5.1. Добавим **delay** из категории **функции** и соединим его с блоком **inject**, параллельно блоку **mqtt out**. Установим задержку в размере 1 секунды.



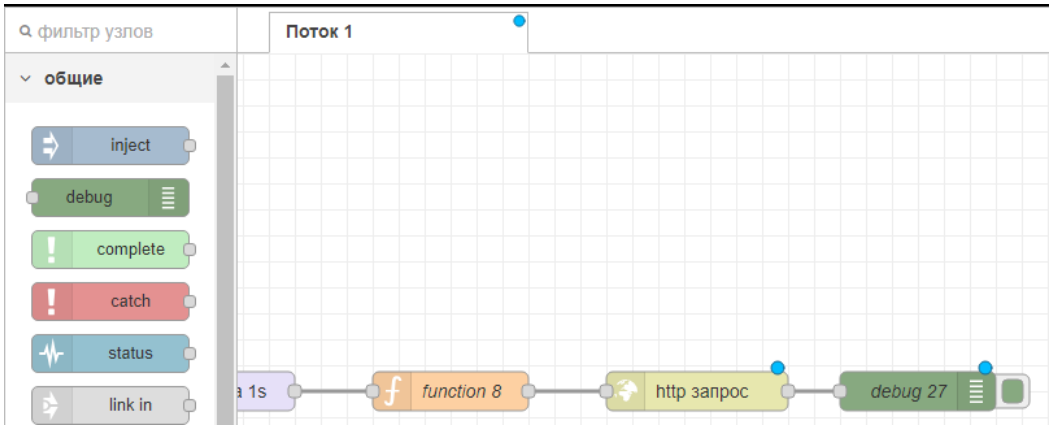
5.2. Последовательно подключим блок **function** из категории **функции**. Зададим ему следующую функцию:

```
const lic_hi = 9008;
const lic_lo = 51717;

let jsonObj = {
    "password":
"5994471abb01112afcc18159f6cc74b4f511b99806da59b3caf5a9c173cacfc5",
    "request_type": "subscribe",

    "licenseID_hi": lic_hi,
    "licenseID_lo": lic_lo,

    "word": {
        "range_begin": 400,
        "range_end": 800,
        "list": [333, 334, 335, 336, 340, 344, 348, 349, 351, 353, 358, 359,
367, 376, 378, 380, 381]
    }
}
msg.payload = jsonObj;
return msg;
```

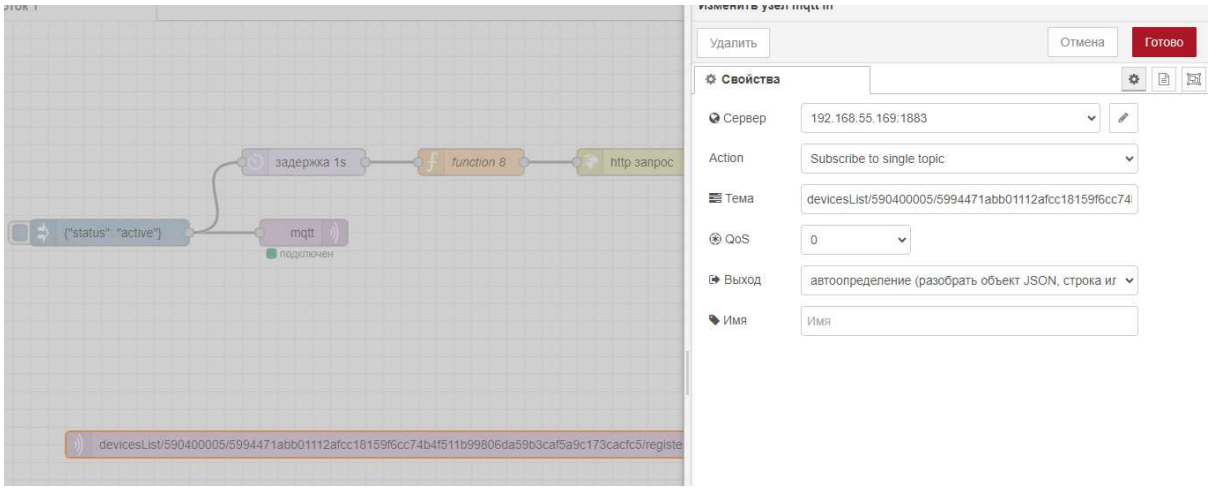
5.3. Добавим последовательно блок **http request** из раздела **сеть**. Зададим URL (адрес EasyHomeCloud и порт HTTP сервера 1500) + **/api/v1/device_request**



5.4. Далее подключим блок **debug** из раздела **общие** для вывода ответов в правую часть интерфейса.

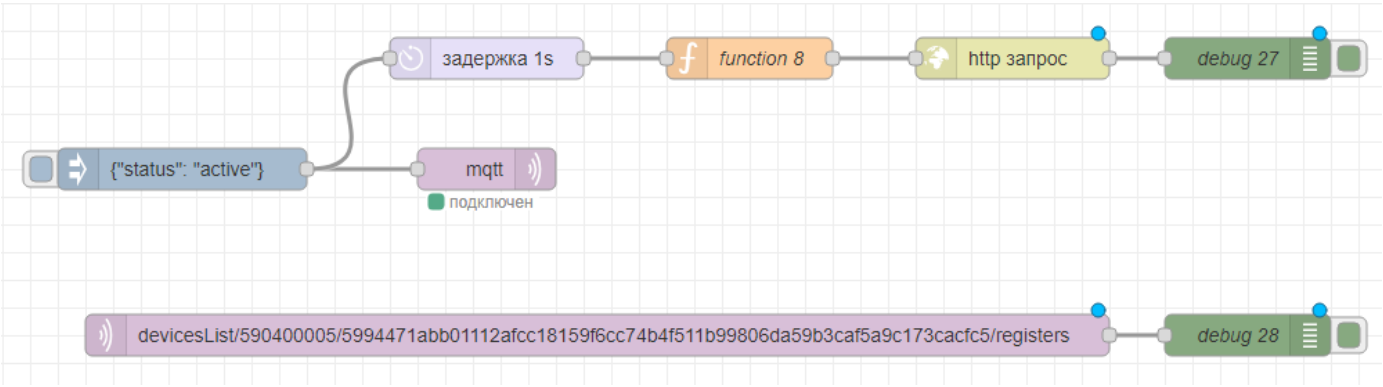


6. Создадим модуль для приёма mqtt сообщений от брокера по подписке.
   6.1. Добавим в пустое пространство этого же потока (вкладки) блок **mqtt in** из раздела **сеть**. Выставим настройки следующим образом:
      a. Выберем тот же сервер, что и ранее
      b. Подпишемся на один топик – топик для регистров конкретного устройства: **devicesList/*<ПЛК ID>*/*<passwordHash>*/registers**

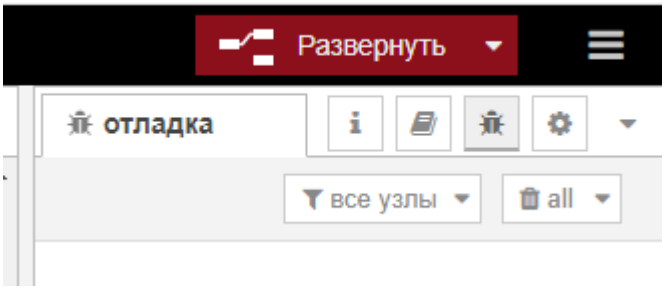


      c. Добавим блок **debug** для вывода приходящих сообщений от брокера.

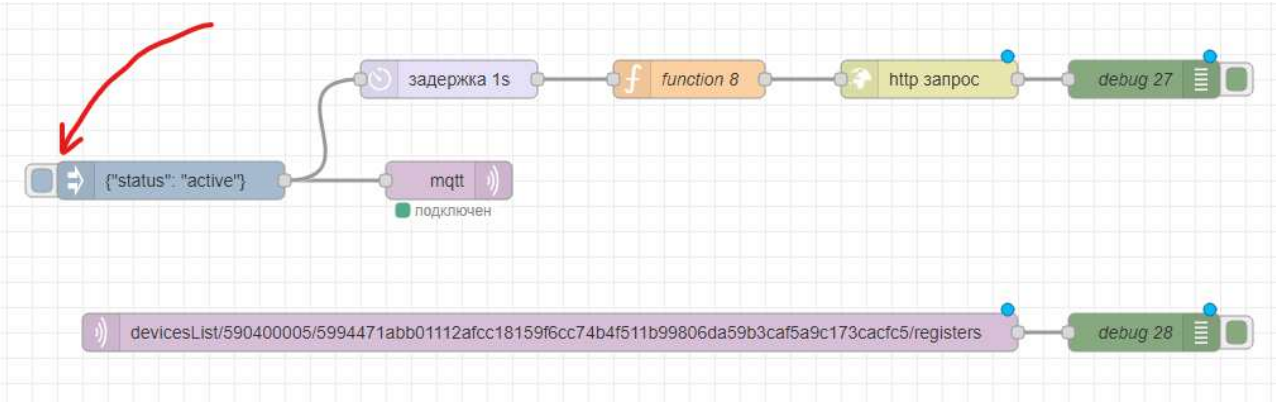
В результате получим программу:



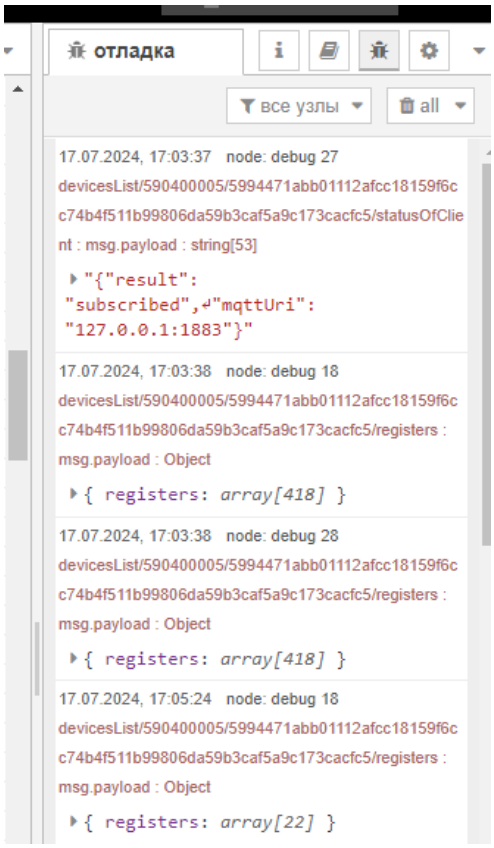
7. Развернём поток с помощью соответствующей кнопки в правом верхнем углу:



8. Выполним эмуляцию клиента (отправим http-запрос с подпиской на mqtt-топик регистров устройства) с помощью нажатия на кнопку блока **inject**:



В результате, в правом окне **отладка** будут выводиться все пришедшие сообщения:



7. Настройки в settings.ini

[EasyhomeProtocol]
KEEP_ALIVE_SEC=120
//Время жизни соединения (с клиентом или устройством)

CASH_LIFE_TIME_MS=1000
//Время жизни кэша (данных о регистрах), мс.

PERIODICAL_REQUEST_INTERVAL_MS=5100
//Частота периодического запроса, чтобы ПЛК не отключил сервер

TIMER_CASH_WAITING=200
//Время ожидания кэша, мс. Для исключения повторного запроса с такими же регистрами на ПЛК.

MQTT_BLOCK_MAX_SIZE=700
//Максимальное количество регистров в одном запросе на ПЛК

MQTT_BLOCK_INTERVAL_MS=200
// Время ожидания между отправками запросов на ПЛК, мс.

[MQTT]
//Если заголовок присутствует, то модуль запускается с указанными в блоке настройками, либо с настройками по-умолчанию. Если заголовок отсутствует – модуль не запускается.

MQTT_BROKER_URI=127.0.0.1:1883
//Адрес брокера

MQTT_NAME=easyhomecloud
//Имя текущего клиента брокера (задается в настройках брокера)

MQTT_PASS=easyhomecloud
//Пароль текущего клиента брокера (задается в настройках брокера)

DATA_UPD_PERIOD=850
//Период опроса всех регистров всех контроллеров для выявления новых данных, мс

MESS_READ_PERIOD=1000
//Период чтения MQTT топика с информацией о подписанных на устройства клиентах, мс.

CLIENTS_LIFETIME=15000000
// Время жизни подписанных на устройства клиентов, мс. При отправке в топик устройства (devicesList/*<ПЛК ID>*/*<passwordHash>*/statusOfClient) сообщения об активности от любого клиента этот таймер запускается для данного устройства. По истечении таймера список регистров для мониторинга у этого устройства очищается

REGS_LIFETIME=15000000
// Время жизни конкретного регистра для каждого устройства, мс. По истечении времени регистр удаляется из списка для мониторинга. Обновляется таймер при подписке любого клиента на конкретный регистр (в том числе, в составе диапазона регистров).