Руководство пользователя ПО “WebboMonitor”

1.Введение

1.2 О программном обеспечении

Программное обеспечение "WebboMonitor" является программным обеспечением для эффективного управления и мониторинга майнинговой инфраструктуры, обеспечивая централизованный контроль над всеми аспектами работы ASIC-устройств. Программное обеспечение позволяет собирать данные с устройств для майнинга криптовалют (далее ASIC-майнеры) и приборов учета электроэнергии (далее счетчики) для дальнейшего анализа. Помимо сбора данных ПО "WebboMonitor" предоставляет программный интерфейс для управления ASIC-майнерами (далее API) и редактирования основных настроек самого модуля.

1.3 Системные требования

Для установки программного обеспечения целевая система должна иметь постоянный доступ к сети Интернет.

Рекомендуемая операционная система Debian 12 "Bookworm" (или Ubuntu 24.04 "Noble Numbat") , распространяемые под свободными лицензиями.

	Минимальные	**Рекомендуемые**
Процессор	1 гигагерц (ГГц) с двумя ядрами	2 гигагерц (ГГц) с четырьмя ядрами
Оперативная память	4 Гб RAM	8 Гб RAM или выше
Свободное место на SSD	128 Гб	128 Гб или больше

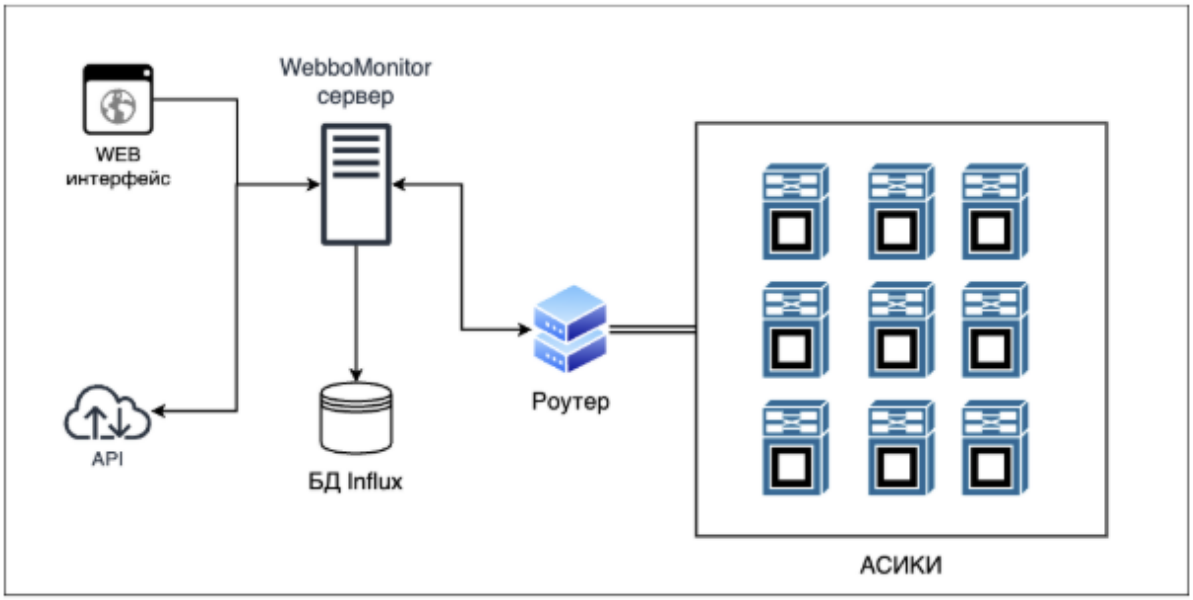
Для сбора данных с приборов учета электроэнергии необходим преобразователь USB-TTL/RS485.

1.4 Основные компоненты

1. ASIC-майнеры: Это специализированные устройства, которые предназначены для выполнения конкретных задач, связанных с майнингом криптовалют. Они оптимизированы для выполнения хеш-функций, таких как SHA-256 для Bitcoin. ASIC-майнеры способны выполнять миллиарды вычислений в секунду, что делает их высокоэффективными в процессе добычи криптовалюты.
2. Локальная сеть: Все ASIC-устройства подключены к роутеру и работают в одной локальной сети. Это позволяет им взаимодействовать друг с другом и с внешними системами. Каждый майнер получает уникальный IP-адрес, что упрощает их идентификацию и управление.
3. Веб-приложение: приложение (программное обеспечение “WebboMonitor”) сканирует локальную сеть для сбора данных о состоянии и характеристиках устройств. Оно предоставляет пользователю интерфейс для мониторинга производительности ASIC-майнеров в реальном времени и управления ими через веб-интерфейс.
4. API (Application Programming Interface): API открывает доступ к функциям управления и мониторинга ASIC-устройств. Он позволяет интегрировать данные с различных устройств и отправлять команды на выполнение определенных действий (например, перезагрузка или изменение настроек) через стандартные HTTP-запросы.

1.5 Принцип работы

Оборудование (размещаемое на различных площадках) может включать значительное количество ASIC-устройств, соединенных в единую локальную сеть через маршрутизатор. Сеть сканируется программным обеспечением (далее также – приложение, ПО) **“WebboMonitor”**, которое осуществляет сбор данных с этих устройств с последующим сохранением их в БД InfluxDB (Apache License 2.0, https://github.com/influxdata/influxdb/blob/main/LICENSE-APACHE, https://github.com/influxdata/influxdb). Приложение предоставляет доступ к веб-интерфейсу для мониторинга состояния устройств и их характеристик, а также включает API для взаимодействия с устройствами.



Площадка

1.5.1 Опрос ASIC-майнеров

Если упрощать, то опрос каждого IP-адреса сводится к следующим шагам:

1. Осуществляется проверка существования устройства с заданным IP-адресом во внутреннем списке зарегистрированных устройств (иными словами, нет ли устройства в кэше). Если устройство с заданным IP-адресом не найдено, то осуществляется переход к шагу 2, в противном случае переходит к шагу 3.
2. Осуществляется попытка установки связи с заданным IP-адресом и идентифицировать модель устройства. В случае успеха осуществляется переход к шагу 3, в противном случае опрос данного IP-адреса прекращается на минуту.
3. Осуществляется попытка получения всех необходимых данных о работе устройства (описаны в следующем разделе) с использованием API, например, cgminer API или совместимого с ним.
4. Парсинг полученных данных, валидация, стандартизация и последующая их запись в InfluxDB.

1.5.2 Управление ASIC-майнерами

Управление ASIC-майнерами реализуется через протоколы HTTP или SSH в зависимости от производителя и модели каждого устройства. Под управлением подразумевается смена режимов работы устройств, перезагрузка, установка майнинг- пулов, управление световыми индикаторами устройств для поиска их физического местоположения в помещениях. Для всех перечисленных операций необходимы соответствующие методы API, которые доступны по протоколу HTTP.

1.5.3 Опрос счетчиков

Помимо опроса ASIC-майнеров требуется сбор данных со счетчиков Меркурий от группы компаний «ИНКОТЕКС». Для этого требуется корректное подключение и возможность добавления счетчиков через API. Осуществляется последовательный опрос добавленных счетчиков с использованием протокола обмена описанного на сайте производителя, после чего записывает полученные данные в InfluxDB.

2. Начало работы

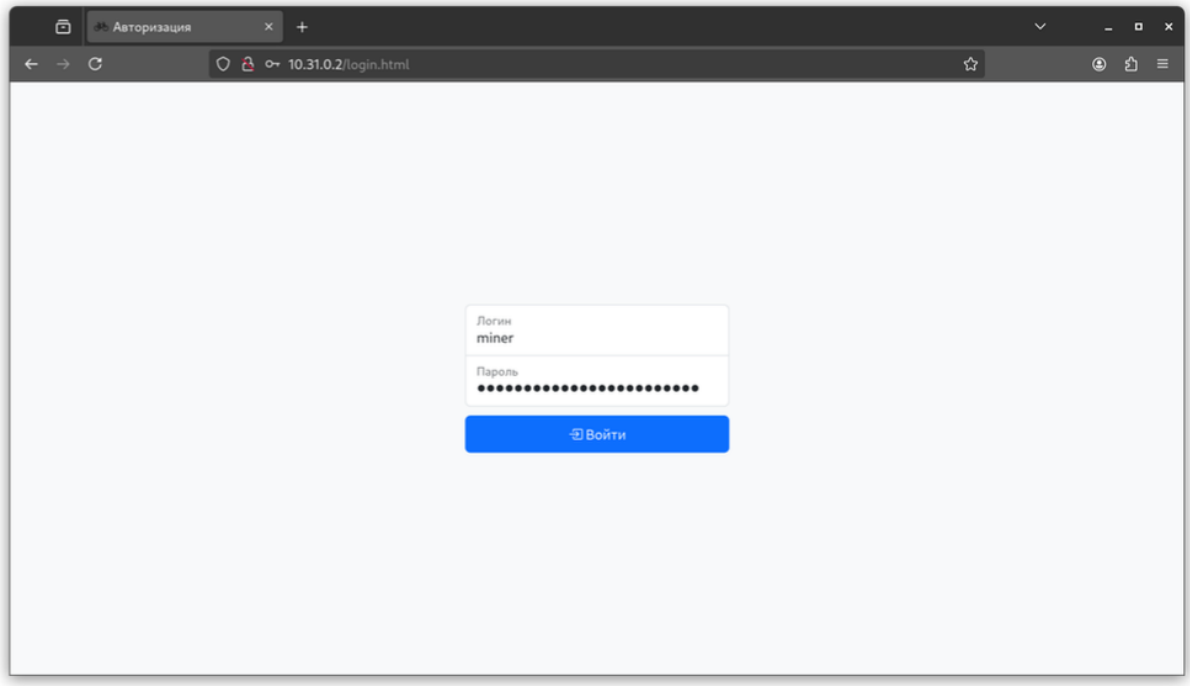
После установки станет доступен веб-интерфейс приложения по адресу:

- http://10.31.0.2/

Данный адрес необходимо открыть в веб-браузере и авторизоваться используя следующие данные:

имя пользователя: miner

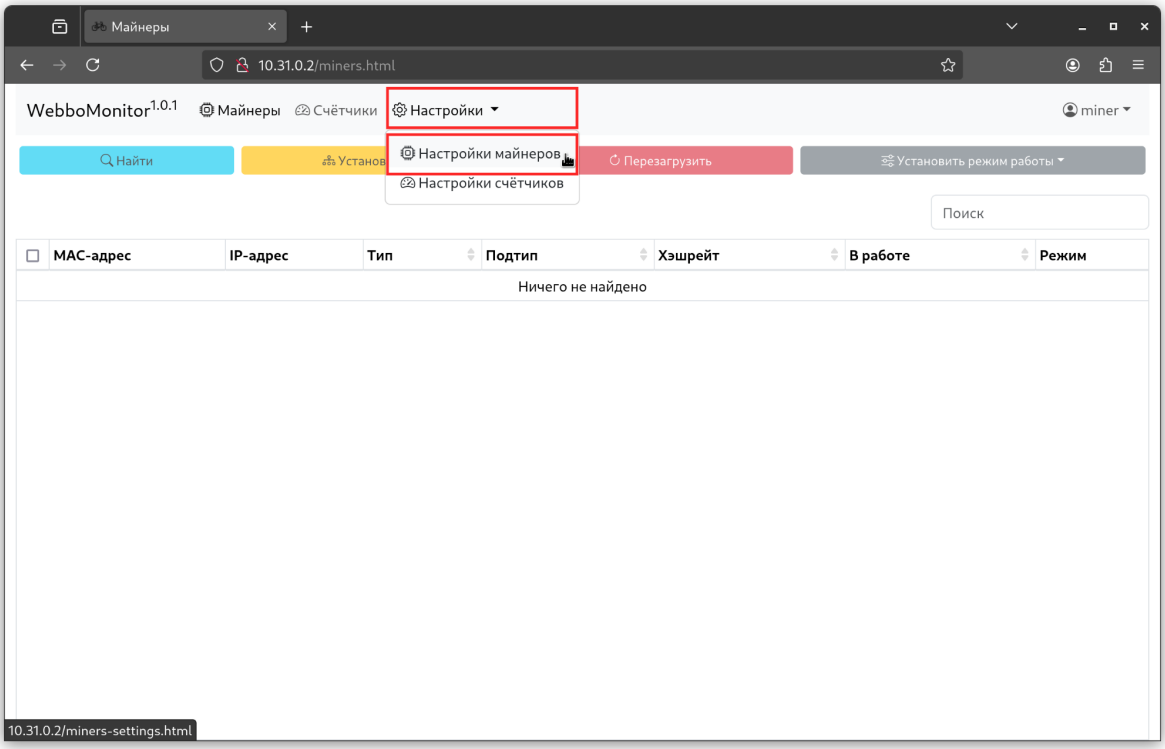
пароль: DqHFY2XOU7ecEuEdBSjJRIIn



2.1 Настройка диапазонов IP-адресов

После авторизации вы будете перенаправлены на страницу с таблицей, содержащей информацию об устройствах для майнинга. Сразу после установки таблица будет пустой, поскольку ещё не заданы диапазоны IP-адресов для сканирования сети.

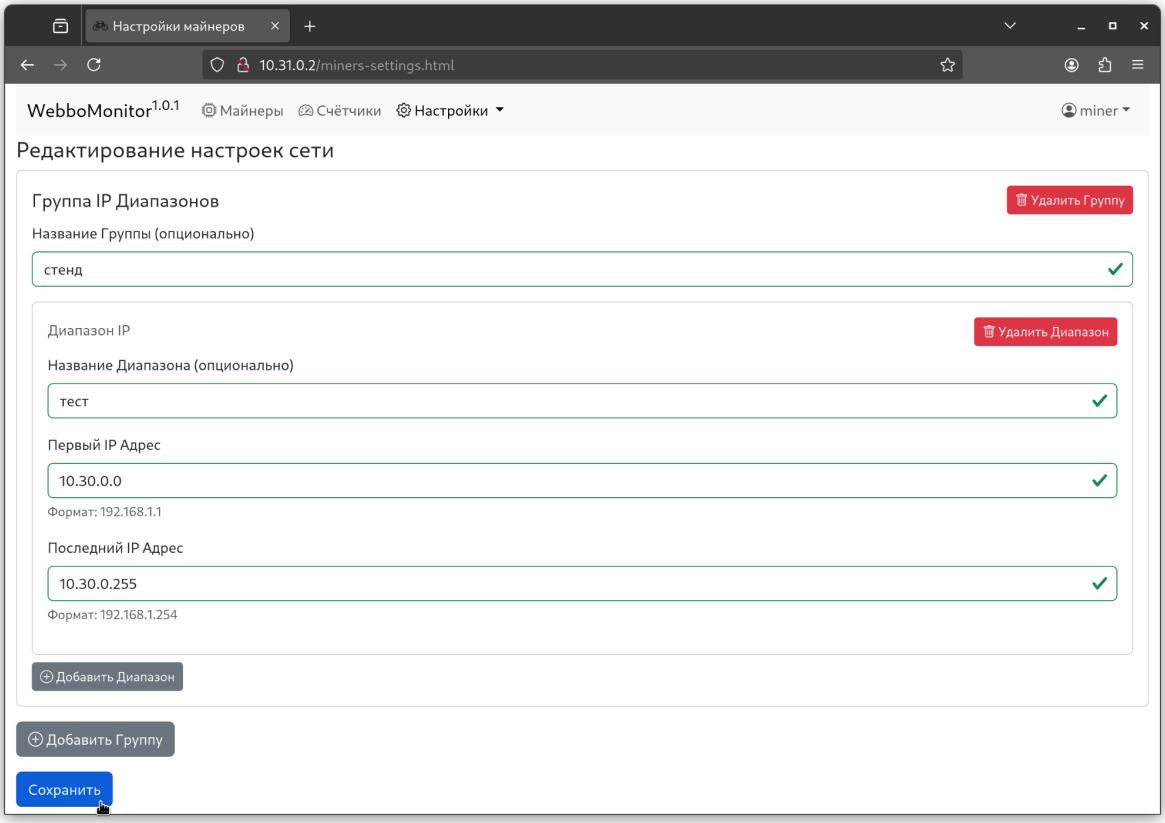
Чтобы задать их, в навигационном меню в верхней части страницы нажмите на пункт **Настройки ,** затем в выпадающем списке выберите **Настройки майнеров .**



Страница редактирования диапазонов IP-адресов содержит форму, позволяющую добавлять группы диапазонов и сами диапазоны адресов, назначенных майнинг-устройствам.

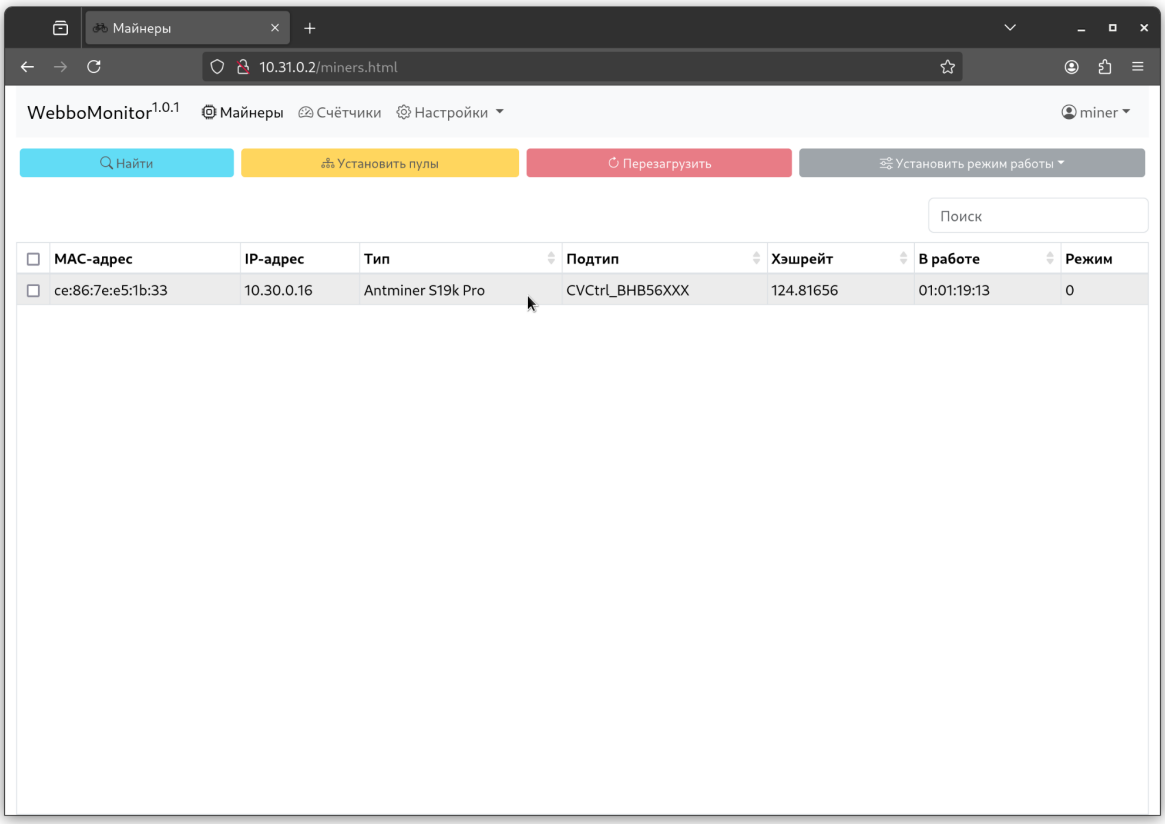
Группы диапазонов могут содержать один и более диапазон. Под группами подразумеваются помещения, а под диапазонами стеллажи (требуется дополнительная настройка сетевого оборудования, что выходит за рамки данного руководства).

В большинстве случаев достаточно одного диапазона в одной единственной группе. Тестовая площадка содержит устройства в диапазоне от 10.30.0.0 до 10.30.0.255 . Этот диапазон и нужно указать, после чего нажать на кнопку **Сохранить .**



Для добавления новой группы диапазонов необходимо нажать кнопку **Добавить группу .** Для удаления группы необходимо нажать на кнопку **Удалить группу.** Для добавления диапазона в группу необходимо нажать на кнопку **Добавить диапазон**, для удаления нажать на кнопку **Удалить диапазон.** После редактирования и нажатия на кнопку **Сохранить** начнется процесс опроса майнинг-устройств в указанных диапазонах.

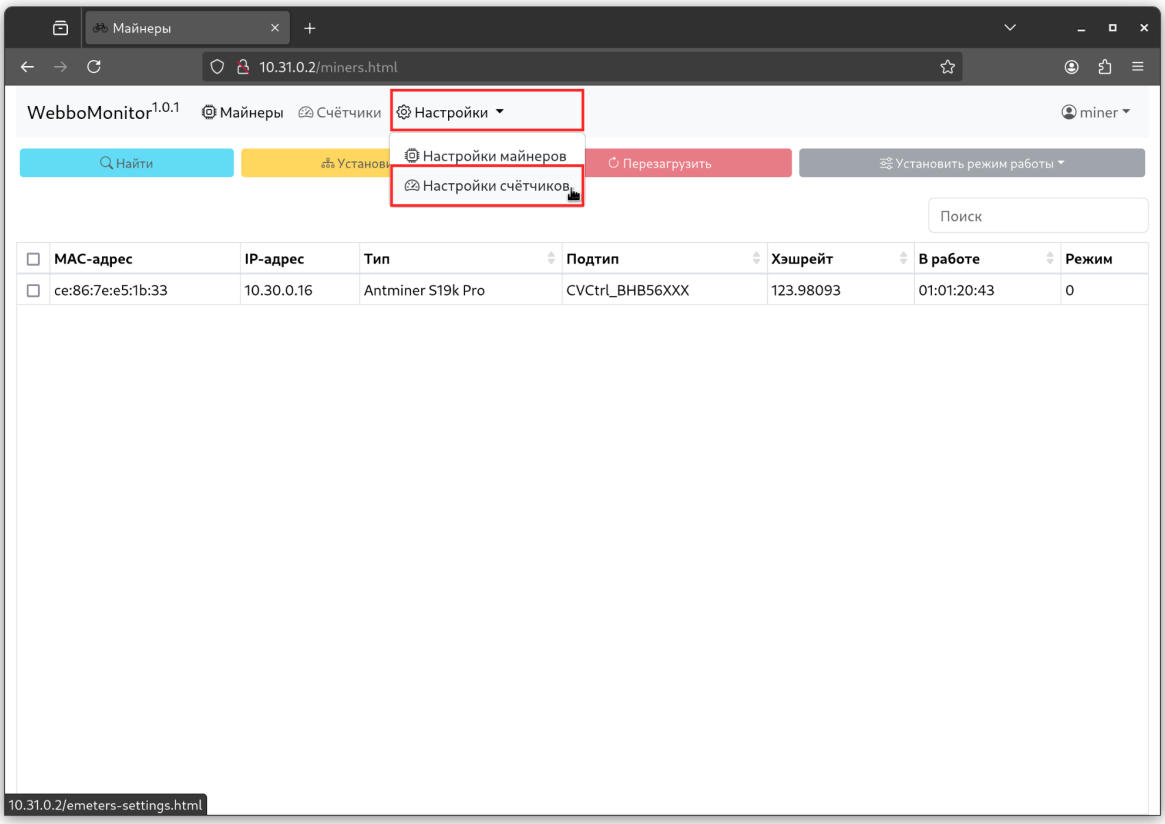
После настройки диапазонов IP-адресов можно вернуться на предыдущую страницу нажав пункт **Майнеры** в навигационном меню. Если в указанных диапазонах найдутся устройства, они будут отображены в таблице.



2.2 Добавление счетчиков электроэнергии

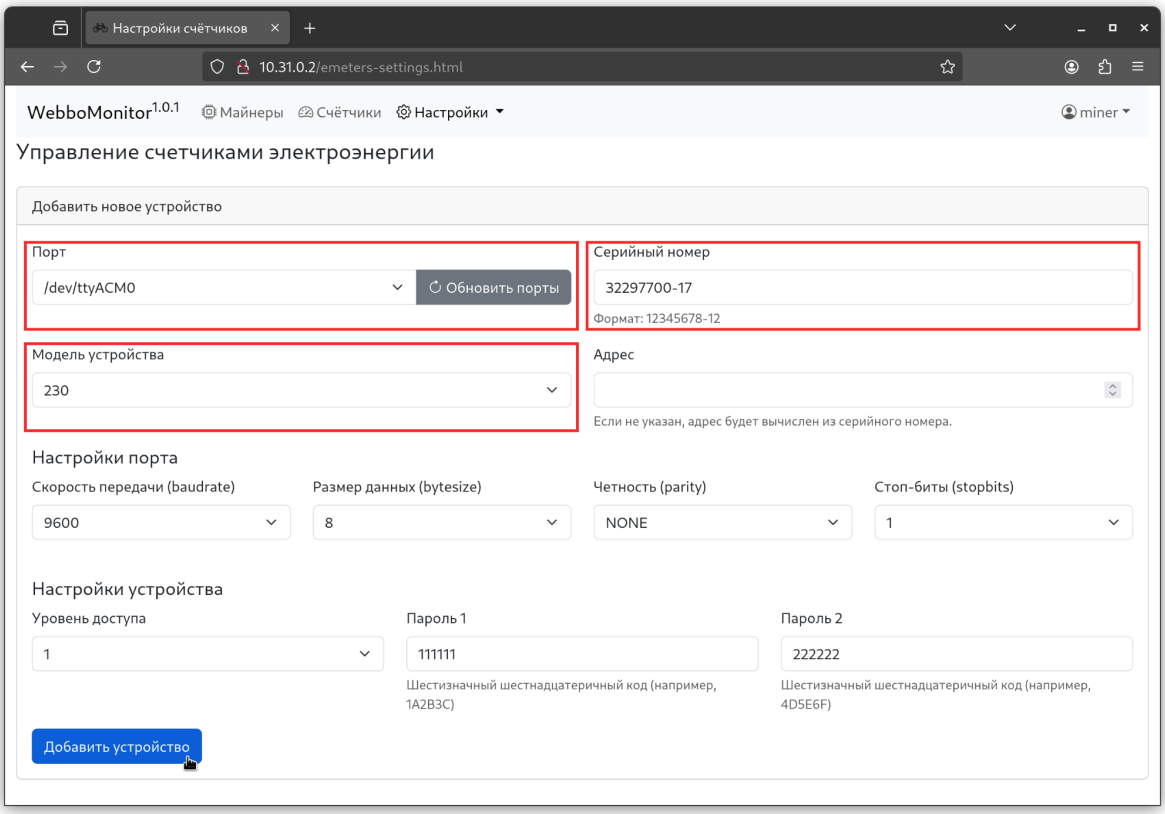
Поддерживается работа с приборами учета электроэнергии Меркурий от группы компаний «ИНКОТЕКС» посредством их опроса. Для подключения счетчиков необходим поддерживаемый операционной системой GNU Linux преобразователь RS485-USB. Инструкция по подключению и конфигурированию доступна на сайте производителя. Для некоторых моделей устройств поддерживающих протокол СПОДЭС может понадобиться переключение протокола со СПОДЭС на Меркурий.

После корректного подключения счетчика необходимо добавить счетчик в систему. Для этого необходимо в верхней части страницы нажать на пункт меню **Настройки**, после чего в выпадающем списке нажать на пункт меню **Настройки счетчиков**.



Страница работы со счетчиками состоит из формы добавления нового счетчика и таблицы, содержащей уже добавленные в систему счетчики.

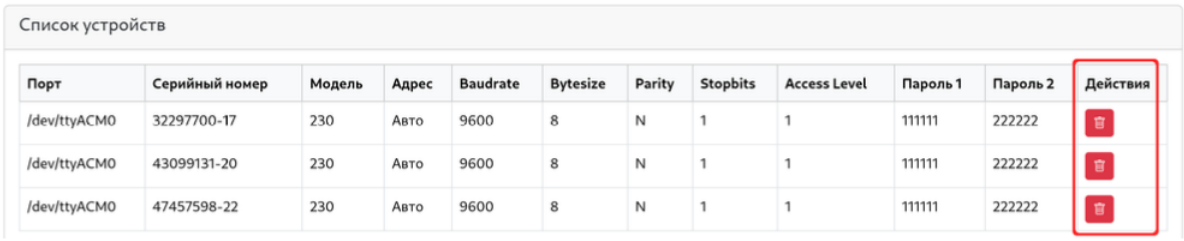
Для добавления счетчика в систему необходимо нажать на кнопку **Обновить порты**, после чего выбрать порт, через который подключен счетчик. Далее необходимо указать модель и серийный номер прибора.



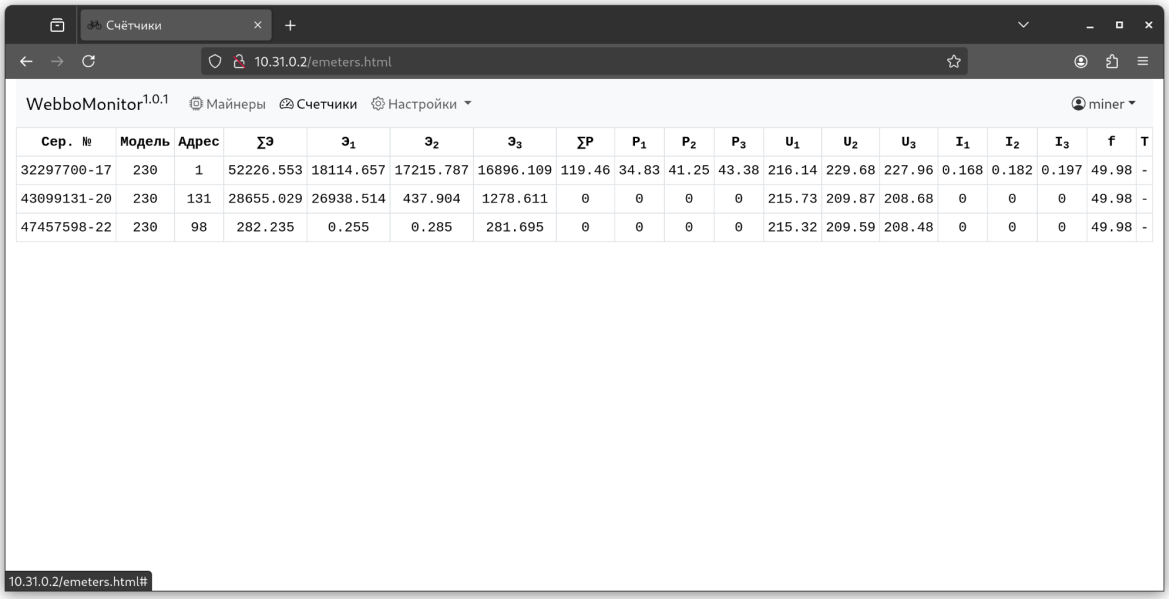
Настройки порта в большинстве случаев менять не нужно. В случаях, когда при конфигурировании прибора был поменян его адрес или пароли, их необходимо указать. После указания всех необходимых данных следует нажать на кнопку **Добавить устройство**. Если все указано верно и прибор отвечает на тестовые запросы, он будет добавлен в систему.

В качестве примера на скрине выше: к тестовому стенду подключено через порт **`/dev/ttyACM0`** три счетчика ***Меркурий 230*** со следующими серийными номерами: **`32297700-17`**; **`43099131-20`**; **`47457598-22`**. Скорость передачи **`9600`**.

Для удаления счетчика из системы необходимо найти его в списке устройств и нажать на кнопку удаления.



После добавления счетчиков можно переместиться на страницу содержащую таблицу с данными, которые забираются со счетчиков. Для этого необходимо кликнуть по пункту **Счетчики** в навигационном меню.



Сер. №	Модель	Адрес	∑Э	$Э_1$	$Э_2$	$Э_3$	∑P	P_1	P_2	P_3	U_1	U_2	U_3	I_1	I_2	I_3	f	T
32297700-17	230	1	52226.553	18114.657	17215.787	16896.109	119.46	34.83	41.25	43.38	216.14	229.68	227.96	0.168	0.182	0.197	49.98	-
43099131-20	230	131	28655.029	26938.514	437.904	1278.611	0	0	0	0	215.73	209.87	208.68	0	0	0	49.98	-
47457598-22	230	98	282.235	0.255	0.285	281.695	0	0	0	0	215.32	209.59	208.48	0	0	0	49.98	-

3. API

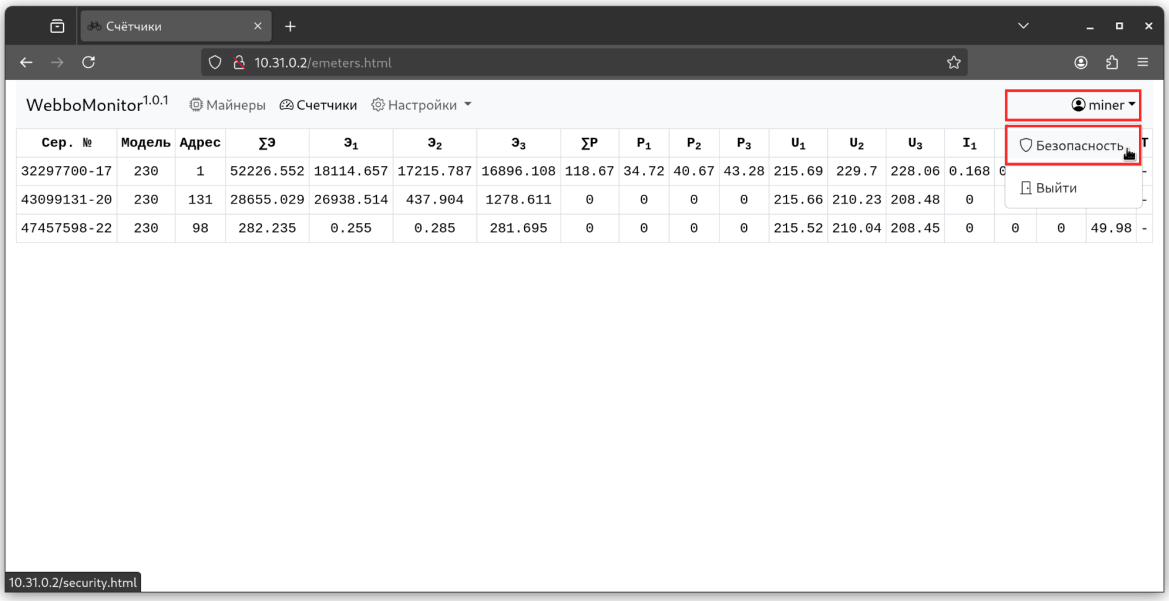
Интерфейс программирования приложений (API) представляет собой набор методов, вызываемых путем отправки **HTTP-запросов** методом **POST**.

HTTP-запрос должен удовлетворять следующим требованиям:

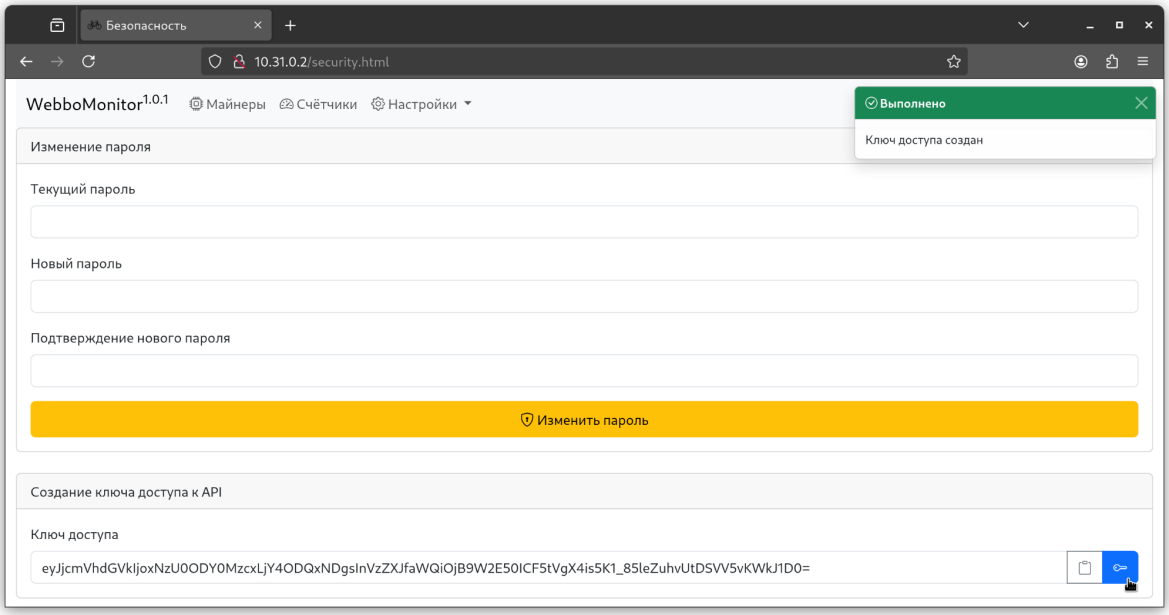
- Параметры передаются в теле запроса в формате JSON, а заголовок запроса должен содержать Content-Type: application/json.
- Запрос должен содержать ключ доступа к API в виде заголовка X-Api-Key.

3.1 Получение ключа доступа к API

Чтобы получить ключ доступа к API, в навигационном меню нажмите на имя пользователя **miner**, затем в выпадающем списке выберите пункт **Безопасность**.



После перехода на страницу **Безопасность** необходимо нажать на кнопку с изображением ключа для создания ключа доступа.



Для копирования ключа доступа в буфер обмена можно кликнуть по кнопке, которая находится левее, либо просто кликнуть правой кнопкой мыши по полю и в появившемся контекстном меню выбрать ***Копировать*****.**

Далее в инструкции будут приведены примеры работы с методами API через утилиту **curl**.
Эти примеры предполагают, что у вас задана переменная окружения **API_KEY**.

Для задания переменной выполните следующую команду в терминале:

API_KEY*='тут_ключ_доступа'*

Например, если использовать ключ доступа с приведенного выше скриншота, получится следующая команда:

API_KEY*='eyJjcmVhdGVkIjoxNzU0ODY0MzcxLjY4ODQxNDgsInVzZXJfaWQiOjB9W2E50ICF5tVgX4is5K1_85leZuhvUtDSVV5vKWkJ1D0='*

3.2 Структура ответов API

Ответы от API всегда имеют следующую структуру:

```
from typing import TypedDict

class ApiResponse(TypedDict):
    success: bool
```

payload: object

То есть содержат два поля **success** и **payload**.

При возникновении ошибки поле **success** принимает значение **false**, а поле **payload** будет содержать строку с описанием ошибки. Например,

```
{
   "success": false,
   "payload": "Something is wrong!"
}
```

В случае успеха поле **success** принимает значение **true**, а поле **payload** будет содержать результат выполнения метода. Например,

```
{
   "success": true,
   "payload": {
      "foo": "bar"
   }
}
```

3.3 Виды методов API

В API существуют два вида методов:

1. Методы, которые выполняются сразу и возвращают результат непосредственно в ответе.
2. Методы, при вызове которых создается задача, и возвращается её идентификатор.

Если с первым видом всё понятно, то второй заслуживает отдельного внимания. Такие методы создают задачу и возвращают её идентификатор — **task_id**.

Пример успешного ответа:

```
{
  "success": true,
  "payload": {
    "task_id": "2cd570ad-add4-4c45-bde4-0089dc856c69"
  }
}
```

Имея идентификатор задачи **task_id**, можно отслеживать её статус, вызывая метод **/api/task-info**:

```
POST /api/task-info HTTP/1.1
Content-Type: application/json
Content-Length: 53

{ "task_id": "22d142e3-4a35-4be0-bc0d-e89951c056a5" }
```

Пример успешного ответа:

```
HTTP/1.1 200 OK
Content-Type: application/json
Content-Length: 85

{
  "success": true,
  "payload": {
    "status": "success",
    "result": true
  }
}
```

Метод **/api/task-info** принимает идентификатор задачи **task_id** и возвращает объект с полями **status** и **result**.
Поле **status** может принимать следующие значения:

- **invalid** – переданный **task_id** не прошел валидацию (не является UUID4);
- **unknown** – задачи с таким **task_id** нет в системе;
- **success** – задача успешно выполнена;
- **failure** – при выполнении задачи произошла ошибка;
- **pending** – задача ещё выполняется;
- **canceled** – задача была отменена.

Поле **result** обычно содержит **null**, за исключением случаев, когда статус равен:

- **success** — **result** содержит результат выполнения задачи;
- **failure** — **result** содержит описание возникшей ошибки.

Если статус задачи — **pending**, необходимо повторять запрос с некоторым интервалом, пока задача не завершится. В остальных случаях задача считается завершенной, и при повторном запросе вернётся статус **unknown**.

3.4 API для работы с мониторингом майнинга

3.4.1 Объекты

Методы этого подраздела оперируют объектами типа **Network**. На листинге ниже можно ознакомиться с его структурой.

```
from ipaddress import IPv4Address
from uuid import uuid4

from pydantic import UUID4, BaseModel, Field

class IPRange(BaseModel):
    name: str = Field(default='')
    uuid: UUID4 = Field(default_factory=uuid4)
    first: IPv4Address
    last: IPv4Address


class IPRangeGroup(BaseModel):
    name: str = Field(default='')
    uuid: UUID4 = Field(default_factory=uuid4)
    ip_ranges: list[IPRange] = Field(default_factory=list)


class Network(BaseModel):
    ip_range_groups: list[IPRangeGroup] = Field(default_factory=list)
```

Объекты типа **Network** имеют единственное поле **ip_range_groups**, который содержит список объектов **IPRangeGroup**. Объекты **IPRangeGroup** по задумке представляют из себя помещения и содержат списки диапазонов сети для мониторинга (списки объектов **IPRange**).

3.4.2 Получение информации об опрашиваемых диапазонах сети

Метод **/api/mining/get-network** служит для получения информации об опрашиваемых группах диапазонов локальной сети. Этот метод не принимает

параметров, следовательно нужно прислать тело запроса содержащее пустой объект **{}**. Результат возвращается сразу.

Пример выполнения запроса:

```
curl -s -X POST \
      -H "X-Api-Key: ${API_KEY}" \
      -H "Content-Type: application/json" \
      --data-raw "{}" \
      http://10.31.0.2/api/mining/get-network | jq .
```

Пример ответа:


```
{
  "success": true,
  "payload": {
    "ip_range_groups": [
      {
        "name": "стенд",
        "uuid": "2a67feee-08f0-4ce1-89cd-7f8282bd1369",
        "ip_ranges": [
          {
            "name": "mecm",
            "uuid": "7c1cfbf8-76b5-41e1-a61e-6dcb5fb40dbd",
            "first": "10.30.0.0",
            "last": "10.30.0.255"
          }
        ]
      }
    ]
  }
}
```


3.4.3 Установка групп опрашиваемых диапазонов

Метод **/api/mining/set-network** служит для установки опрашиваемых групп диапазонов локальной сети. Принимает объект типа **Network**. Если переданный объект проходит валидацию, то возвращается идентификатор задачи **task_id**.

Пример запроса с помощью curl:

```
NETWORK=$(cat <<'EOF'
{
    "ip_range_groups": [
     {
       "name": "подсобка",
       "ip_ranges": [
        {
          "name": "пятая стойка",
          "first": "10.30.0.0",
          "last": "10.30.0.128"
        }
       ]
     }
   ]
}
EOF
)
curl -s -X POST \
       -H "X-Api-Key: ${API_KEY}" \
       -H "Content-Type: application/json" \
       --data-raw "${NETWORK}" \
       http://10.31.0.2/api/mining/set-network | jq .
```

Пример ответа:

```
{
  "success": true,
  "payload": {
    "task_id": "ede2229f-d4d0-4118-a51c-543a07cb493a"
  }
}
```

Как видно метод вернул идентификатор задачи **task_id**.

3.4.4 Получение статуса выполнения задачи

Далее с помощью метода **/api/task-info** можно получить статус выполнения интересующей задачи. В следующем примере переменной **TASK_ID** нужно присвоить ваш собственный идентификатор задачи, полученный в ответе на предыдущий запрос:

```
TASK_ID="ваш task_id из ответа"
```

```
curl -s -X POST \
      -H "X-Api-Key: ${API_KEY}" \
      -H "Content-Type: application/json" \
      --data-raw "{\"task_id\":\"${TASK_ID}\"}" \
      http://10.31.0.2/api/task-info | jq .
```

Для приведенного выше примера **task_id** имеет значение **ede2229f-d4d0-4118-a51c-543a07cb493a**. Таким образом выполняем:

```
TASK_ID="ede2229f-d4d0-4118-a51c-543a07cb493a"

curl -s -X POST \
      -H "X-Api-Key: ${API_KEY}" \
      -H "Content-Type: application/json" \
      --data-raw "{\"task_id\":\"${TASK_ID}\"}" \
      http://10.31.0.2/api/task-info | jq .
```

Пример ответа:

```
{
  "success": true,
  "payload": {
    "status": "success",
    "result": true
  }
}
```

В случае успешного выполнения задача возвращает значение **true**.

В дальнейшем в руководстве примеры запросов для получения статуса задач будут опущены. Для удобства и наглядности будут приводиться только итоговые результаты выполнения.

3.5 ASIC майнеры

3.5.1 Основные параметры

При вызове методов из этого раздела **обязательно** указывается **только один** из перечисленных ниже взаимоисключающих параметров:

- **macs** – список MAC-адресов устройств, к которым применяется метод. При необходимости применения метода **ко всем** устройствам, известным системе, укажите `**`.

- **ip_range** – диапазон IP-адресов (объект типа **IPRange**, содержащий поля **first** и **last**), к устройствам в котором применяется метод.

- **ip_range_id** – идентификатор объекта **IPRange**, к устройствам в котором применяется метод.

- **ip_range_group_id** – идентификатор объекта **IPRangeGroup**, к устройствам в котором применяется метод.

Ниже приведен псевдокод для наглядности:

```
from ipaddress import IPv4Address
from typing import Annotated, Literal
from uuid import UUID, uuid4

from pydantic import UUID4, BaseModel, Field

type MACAddr = Annotated[
    str, Field(pattern=r'^[0-9a-f]{2}(?:\:[0-9a-f]{2}){5}$')
]


class IPRange(BaseModel):
    name: str = Field(default='')
    uuid: UUID4 = Field(default_factory=uuid4)
    first: IPv4Address
    last: IPv4Address


class MinersAPIBaseParams(BaseModel):
    macs: Literal['*'] | list[MACAddr] | None = None
    ip_range: IPRange | None = None
    ip_range_id: UUID4 | None = None
    ip_range_group_id: UUID4 | None = None
```

3.5.2 Перезагрузка устройств

Метод `**/api/mining/miners/reboot**` выполняет перезагрузку указанных устройств.

Например, для перезагрузки **всех** устройств:

```
curl -s -X POST \
        -H "X-Api-Key: ${API_KEY}" \
        -H "Content-Type: application/json" \
        --data-raw '{"macs":"*"}' \
```

http://10.31.0.2/api/mining/miners/reboot | jq .

Пример ответа при успешной постановке задачи:

```
{
  "success": true,
  "payload": {
    "task_id": "1363efc2-763c-42b4-8f1b-3df592618245"
  }
}
```

После завершения задачи метод возвращает результат следующего вида:

```
{
  "success": true,
  "payload": {
    "status": "success",
    "result": {
      "ce:86:7e:e5:1b:33": { "success": true, "payload": null }
    }
  }
}
```

В объекте результата:

- **ключи** – MAC-адреса устройств, к которым был применён метод;
- **значения** – результат выполнения метода на устройстве, где поле `payload` содержит текстовое описание ошибки при значении `success` равном *`false`* и результат выполнения метода при значении поля `success` равном *`true`*.

3.5.3 Поиск устройств

В данном подразделе описана работа со светодиодами устройств. Речь идёт о двух светодиодах, одновременное мигание которых используется для поиска физического местоположения устройства в стойке.
Методы принимают или возвращают значения типа **MinerLedStatus**:

```
from enum import IntEnum

class MinerLedStatus(IntEnum):
    NORMAL = 0
    BLINK = 1
```

Значения имеют следующий смысл:

- `**0**` — светодиоды не мигают;
- `**1**` — светодиоды мигают.

3.5.4 Получение статуса светодиодов

Метод **/api/mining/miners/get-led-status** возвращает текущий статус мигания светодиодов для указанных устройств.

В ответе поле **result** содержит **1**, если мигание включено, и **0** — если выключено.

Например, чтобы узнать состояние светодиодов устройства с **MAC**-адресом **ce:86:7e:e5:1b:33**:

```
curl -s -X POST \
      -H "X-Api-Key: ${API_KEY}" \
      -H "Content-Type: application/json" \
      --data-raw '{"macs":["ce:86:7e:e5:1b:33"]}' \
      http://10.31.0.2/api/mining/miners/get-led-status | jq .
```

Получаем результат выполнения задачи следующего вида:

```
{
  "success": true,
  "payload": {
    "status": "success",
    "result": {
      "ce:86:7e:e5:1b:33": { "success": true, "payload": 0 }
    }
  }
}
```

3.5.5 Установка статуса светодиодов

Метод `/api/mining/miners/set-led-status` позволяет включать или выключать мигание светодиодов на указанных устройствах.
Помимо параметров, определяющих список устройств, необходимо передать параметр `status`:

- `**1**` — включить мигание;
- `**0**` — выключить мигание.

Например, чтобы включить мигание светодиодов для устройств в диапазоне адресов `**10.30.0.10 – 10.30.0.128**`:

```
curl -s -X POST \
      -H "X-Api-Key: ${API_KEY}" \
      -H "Content-Type: application/json" \
      --data-raw '{"ip_range":{"first":"10.30.0.10","last":"10.30.0.128"},"status":1}' \
      http://10.31.0.2/api/mining/miners/set-led-status | jq .
```

И получаем **результат выполнения задачи** вида:

```
{
  "success": true,
  "payload": {
    "status": "success",
    "result": {
      "ce:86:7e:e5:1b:33": {
        "success": true,
        "payload": null
      }
    }
  }
}
```

Как видно из результата, в указанном диапазоне найдено одно устройство с MAC-адресом **ce:86:7e:e5:1b:33**, для которого мигание светодиодов было успешно включено.

Метод возвращает значение **null** (**None** в терминах Python) в случае успешного выполнения.

3.5.6 Режимы работы устройств

Методы данного подраздела позволяют управлять режимами работы устройств. Возможные режимы представлены перечислением:

```
from enum import StrEnum, auto

class MinerWorkMode(StrEnum):
    NORMAL = auto()
    SLEEP = auto()
    LOW_POWER = auto()
    UNKNOWN = auto()
```

- **normal** – нормальный режим работы устройства;
- **sleep** – режим сна;
- **low_power** – режим пониженного энергопотребления;
- **unknown** – значение, возвращаемое API, когда режим работы устройства неизвестен.

*Устройства от Bitmain поддерживают режимы **normal** и **sleep** на всех версиях прошивок. Режим **low_power** поддерживается только на относительно новых версиях прошивок.*

Получение режима работы

Метод **/api/mining/miners/get-work-mode** позволяет получить текущие режимы работы заданных устройств.

Например, чтобы получить режимы работы **всех** устройств, выполняем:

```
curl -s -X POST \
      -H "X-Api-Key: ${API_KEY}" \
      -H "Content-Type: application/json" \
      --data-raw '{"macs":"*"}' \
      http://10.31.0.2/api/mining/miners/get-work-mode | jq .
```

И получаем **результат выполнения задачи** вида:

```
{
  "success": true,
  "payload": {
    "status": "success",
    "result": {
      "ce:86:7e:e5:1b:33": {
        "success": true,
        "payload": "normal"
      }
    }
  }
}
```

Как видно из результата, системе известно одно устройство с MAC-адресом **ce:86:7e:e5:1b:33**, которое в данный момент работает в нормальном режиме.

Установка режима работы

Метод **/api/mining/miners/set-work-mode** позволяет задать режим работы устройств с помощью параметра **mode**. Допустимые значения: **normal**, **sleep** и **low_power**.

Например, чтобы перевести все устройства в режим пониженного энергопотребления, выполните команду:

```
curl -s -X POST \
    -H "X-Api-Key: ${API_KEY}" \
    -H "Content-Type: application/json" \
    --data-raw '{"macs":"*", "mode":"low_power"}' \
    http://10.31.0.2/api/mining/miners/set-work-mode | jq .
```

И получаем результат выполнения задачи вида:

```
{
  "success": true,
  "payload": {
    "status": "success",
    "result": {
      "ce:86:7e:e5:1b:33": { "success": true, "payload": null }
    }
  }
}
```

3.5.7 Майнинг-пулы

Методы работы с пулами оперируют объектами типа **MiningPool**. Ниже приведена структура такого объекта:

```
from typing import Literal

from pydantic import BaseModel, Field

class MiningPool(BaseModel):
    address: str
    worker: str
    passwd: str
    suffix_type: Literal['ip', 'no_change', 'empty'] = Field(
        default='no_change', exclude=True,
    )
```

Пояснения к полям объекта **MiningPool**:

- `**address**` — URL майнинг-пула;

- `**worker**` — имя пользователя;

- `**passwd**` — пароль;

- `**suffix_type**` — тип суффикса для воркера (требуется только при установке майнинг-пулов):

  - `**ip**` — добавляет в суффикс два последних байта IP-адреса в формате `123x234`;

  - `**no_change**` (по умолчанию) — оставляет текущий суффикс без изменений;

  - `**empty**` — не добавляет суффикс к воркеру.

В этом разделе под «пулами» понимаются объекты типа `MiningPool`.

Получение списка пулов

Для получения списка майнинг-пулов с каждого интересующего устройства используется метод `**/api/mining/miners/get-pools**`.

Чтобы получить списки пулов с устройств в диапазоне адресов `**10.30.0.10 - 10.30.0.150**`, выполните команду:

```
curl -s -X POST \
  -H "X-Api-Key: ${API_KEY}" \
  -H "Content-Type: application/json" \
  --data-raw '{"ip_range": {"first": "10.30.0.10", "last": "10.30.0.150"}}' \
  http://10.31.0.2/api/mining/miners/get-pools | jq .
```

Результат выполнения задачи будет иметь следующий вид:

```
{
  "success": true,
```

```
  "payload": {
    "status": "success",
    "result": {
      "ce:86:7e:e5:1b:33": {
        "success": true,
        "payload": [
          {
            "address": "stratum+tcp://super-puper.pool:1337",
            "worker": "john-doe.001",
            "passwd": "123456"
          },
          {
            "address": "stratum+tcp://super-puper.pool:31337",
            "worker": "john-doe.001",
            "passwd": "123456"
          },
          {
            "address": "stratum+tcp://super-puper.pool:42",
            "worker": "john-doe.001",
            "passwd": "123456"
          }
        ]
      }
    }
  }
}
```


Установка списка пулов

Для установки списка пулов используется метод `**/api/mining/miners/set-pools**`. Этот метод, помимо одного из основных параметров, принимает дополнительный параметр `**pools**`, который должен быть списком из трёх элементов типа `**MiningPool**` или значений `**None**` (**null**). Значение `**None**` означает, что соответствующий пул, уже установленный на устройстве, следует оставить без изменений.

class SetPoolsParams(MinersAPIBaseParams):
pools: tuple[MiningPool | None, MiningPool | None, MiningPool | None]

Например, если необходимо на устройствах в диапазоне `**10.30.0.10 - 10.30.0.128**` поменять только второй пул используя в качестве суффикса воркера IP-адрес, то шлем запрос запрос следующего вида:

```
PARAMS=$(cat <<'EOF'
{
   "ip_range": {
      "first": "10.30.0.10",
      "last": "10.30.0.128"
   },
   "pools":[
       null,
       {
         "address": "stratum+tcp://super-duper.pool:443",
         "worker": "james",
         "passwd": "654321",
         "suffix_type": "ip"
       },
       null
   ]
}
EOF
)
curl -s -X POST \
   -H "X-Api-Key: ${API_KEY}" \
   -H "Content-Type: application/json" \
   --data-raw "${PARAMS}" \
   http://10.31.0.2/api/mining/miners/set-pools | jq .
```

Результат выполнения задачи будет иметь следующий вид:

```
{
  "success": true,
  "payload": {
    "status": "success",
    "result": {
      "ce:86:7e:e5:1b:33": {
        "success": true,
        "payload": null
      }
    }
  }
}
```

Как видно из результата, в заданном диапазоне есть одно устройство с mac-адресом **ce:86:7e:e5:1b:33** и для него успешно выполнился данный метод.

Поле `payload` содержит значение `**None**` (**null**) поскольку в случае успешного выполнения данный метод ничего не возвращает.

Теперь, если снова запросим список пулов для этого устройства, то получим такой следующий набор пулов:


```
[
  {
    "address": "stratum+tcp://super-puper.pool:1337",
    "worker": "john-doe.001",
    "passwd": "123456"
  },
  {
    "address": "stratum+tcp://super-duper.pool:443",
    "worker": "james.0x16",
    "passwd": "654321"
  },
  {
    "address": "stratum+tcp://super-puper.pool:42",
    "worker": "john-doe.001",
    "passwd": "123456"
  }
]
```


То есть поменялся только второй пул, а первый и третий остались без изменения.

3.5.8 API для работы с мониторингом устройств, подключенных через последовательные порты

Последовательные порты

В этом разделе описывается работа с устройствами, подключенными через последовательные порты. Для взаимодействия с ними используются модули и pySerial-asyncio. Их особенности и параметры будут заметны при ознакомлении с методами, описанными далее.

Получение списка доступных последовательных портов

Для получения списка доступных последовательных портов используется метод **/api/serial/get-ports**, который не принимает параметров. Для вызова метода необходимо отправить HTTP-запрос используя следующую команду:

```
curl -s -X POST \
    -H "X-Api-Key: ${API_KEY}" \
    -H "Content-Type: application/json" \
    --data-raw '{}' \
    http://10.31.0.2/api/serial/get-ports | jq .
```

Результат выполнения задачи будет содержать список последовательных портов:

```
{
  "success": true,
  "payload": {
    "status": "success",
    "result": [
      "/dev/ttyACM0"
    ]
  }
}
```

В случае успешного выполнения метод возвращает список строк — имён доступных последовательных портов, если таковые имеются.

Информация о доступных портах необходима при добавлении новых устройств в систему. Метод получает данные с помощью функции serial.tools.list_ports.comports из библиотеки **pySerial** и по умолчанию возвращает только порты, подключенные через USB.

3.5.9 Устройства

Получение списка зарегистрированных устройств

Для получения списка зарегистрированных устройств используется метод **/api/serial/devices/get**. Этот метод не принимает параметров и возвращает результат сразу.

Для его вызова необходимо отправить запрос командой следующего вида:

```
curl -s -X POST \
    -H "X-Api-Key: ${API_KEY}" \
    -H "Content-Type: application/json" \
    --data-raw '{}' \
    http://10.31.0.2/api/serial/devices/get | jq .
```

Результат вызова метода содержит список зарегистрированных устройств:


```
{
  "success": true,
  "payload": [
    {
      "port": "/dev/ttyACM0",
      "port_settings": {
        "baudrate": 9600,
        "bytesize": 8,
        "parity": "N",
        "stopbits": 1,
        "timeout": null,
        "xonxoff": false,
        "rtscts": false,
        "write_timeout": null,
        "dsrdtr": false,
        "inter_byte_timeout": null
      },
      "serial_number": "32297700-17",
      "device_model": "230",
      "address": 1,
      "access_level": 1,
      "password_1": "111111",
      "password_2": "222222"
    }
  ]
}
```


Добавление устройства

Метод **/api/serial/devices/add** используется для добавления новых устройств. Он принимает параметр **device** — словарь со следующими полями:

- `**port**` – строка, полученная с помощью метода `**/api/serial/get-ports**`;

- `**serial_number**` – строка, содержащая серийный номер добавляемого устройства;

- `**device_model**` – строка, содержащая модель устройства (поддерживаются: `203.2TD`, `204`, `208`, `230`, `231`, `234`, `236`, `238`).

Дополнительно можно указать необязательные поля:

- `**port_settings**` – словарь с параметрами для создания объекта `serial.Serial` (кроме `port` и `exclusive`). Например: `baudrate` (по умолчанию `9600`), `bytesize` (`8`), `parity` (`N`), `stopbits` (`1`), `timeout` (`None`), `xonxoff` (`False`), `rtscts` (`False`), `write_timeout` (`None`), `dsrdtr` (`False`), `inter_byte_timeout` (`None`);

- `**access_level**` – уровень доступа: число `1` (по умолчанию, достаточно для сбора статистики) или `2`;

- `**password_1**` – строка из 6 символов, пароль для уровня доступа `1` (по умолчанию `111111`);

- `**password_2**` – строка из 6 символов, пароль для уровня доступа `2` (по умолчанию `222222`).

***Пример**: добавление устройства Меркурий 230 с серийным номером `43099131-20`, подключенного к порту `/dev/ttyACM0`:*

```
PARAMS=$(cat <<'EOF'
{
   "device": {
      "port": "/dev/ttyACM0",
      "device_model": "230",
      "serial_number": "43099131-20"
   }
}
EOF
)
curl -s -X POST \
   -H "X-Api-Key: ${API_KEY}" \
   -H "Content-Type: application/json" \
   --data-raw "${PARAMS}" \
   http://10.31.0.2/api/serial/devices/add | jq .
```

В случае успеха результат выполнения задачи будет иметь следующий вид:


```
{
  "success": true,
  "payload": {
    "status": "success",
    "result": true
```

```
  }
}
```

Удаление устройства

Метод **/api/serial/devices/delete** позволяет удалить устройство по его серийному номеру. Для удаления устройства с серийным номером **43099131-20** необходимо отправить запрос следующего вида:

```
curl -s -X POST \
    -H "X-Api-Key: ${API_KEY}" \
    -H "Content-Type: application/json" \
    --data-raw '{"serial_number":"43099131-20"}' \
    http://10.31.0.2/api/serial/devices/delete | jq .
```

Результат выполнения задачи будет иметь следующий вид:

```
{
  "success": true,
  "payload": {
    "status": "success",
    "result": true
  }
}
```

Более детальную и подробную документацию API можно обнаружить по пути `**/opt/m0n/docs/API.md**` после установки ПО.