Практическая работа №9 – Знакомство с облачными платформами IoT - Создание моделей и объектов

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Модель объекта контроля - это формализованное представление устройства, подключаемого к платформе. В модели описываются параметры, которые отправляет устройство, и команды управления, которые оно может отработать. Единожды созданная модель может быть использована как для одного, так и для нескольких объектов, если они обладают одинаковым набором считываемых данных и выполняемых функций.

Если у вас есть три одинаковых датчика, передающих температуру и влажность, то нужно создать только **одну** модель, но **три** объекта (для каждого устройства)

Функции модели:

– поднимает уровень абстракции с уровня протоколов и аппаратной реализации до понятных человеку функций устройства, с которыми удобно работать без необходимости разбираться в протоколе. Преобразованные таким образом данные формируются в JSON, вид которого привычен для большинства разработчиков;

– автоматически рендерит интерфейс для визуализации данных от устройства;

– используется для формирования сценариев автоматизации и обработчиков данных.

Карточка модели

1. Имя модели
2. Описание модели
3. Шаблон (используемый протокол передачи данных)
4. Количество объектов с этой моделью. Нажмите на него, чтобы применить фильтр к списку объектов по данной модели

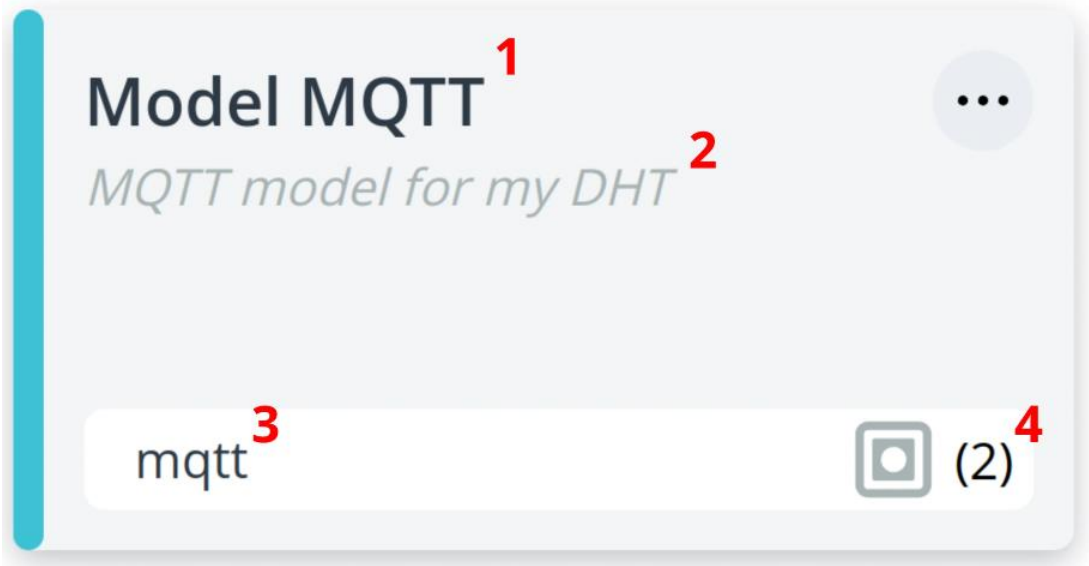


Рисунок. 1. Карточка модели

Создание модели устройства

Для создания новой модели перейдите на вкладку **Модели**. Добавьте модель, нажав на плюсик.

Заполните следующие поля:

- **имя** — наименование модели;
- **описание** — подробная характеристика модели, заполняется при необходимости;
- **импорт** — возможность импортировать готовую модель в виде файла или по ссылке.

Выберите шаблон — заготовленную структуру модели. В платформу добавлено несколько готовых моделей, созданных на основе реализованных в ней протоколов. Модель создается по выбранному шаблону и включает в себя ряд аргументов, событий и действий, которые могут использоваться для устройств с таким протоколом. Вам остается только подстроить модель, добавляя, удаляя и конфигурируя узлы модели.

Также вместо создания новой модели можно воспользоваться одним из стартовых наборов.

Подробнее о стартовых наборах

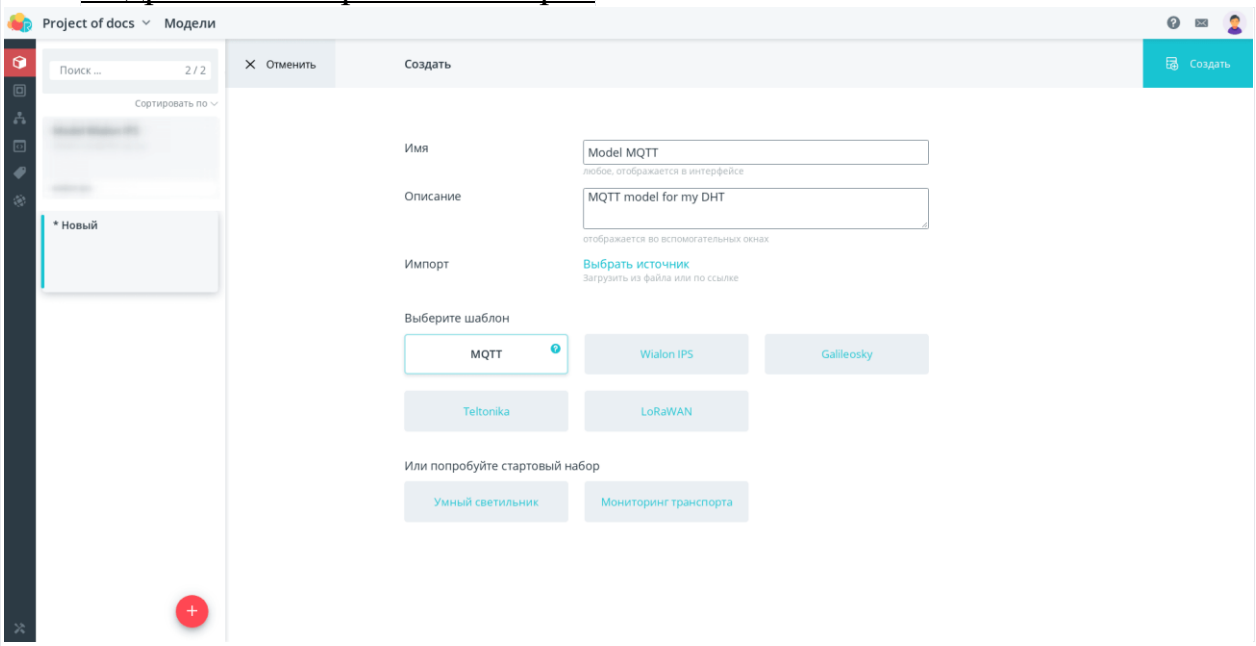


Рисунок 2. Создание модели

Нажмите кнопку **Создать**. Перед вами откроется ваша новая модель. В левой части отображается древовидная структура модели, включающая в себя разнотипные узлы. В правой - окно редактирования выбранного узла.

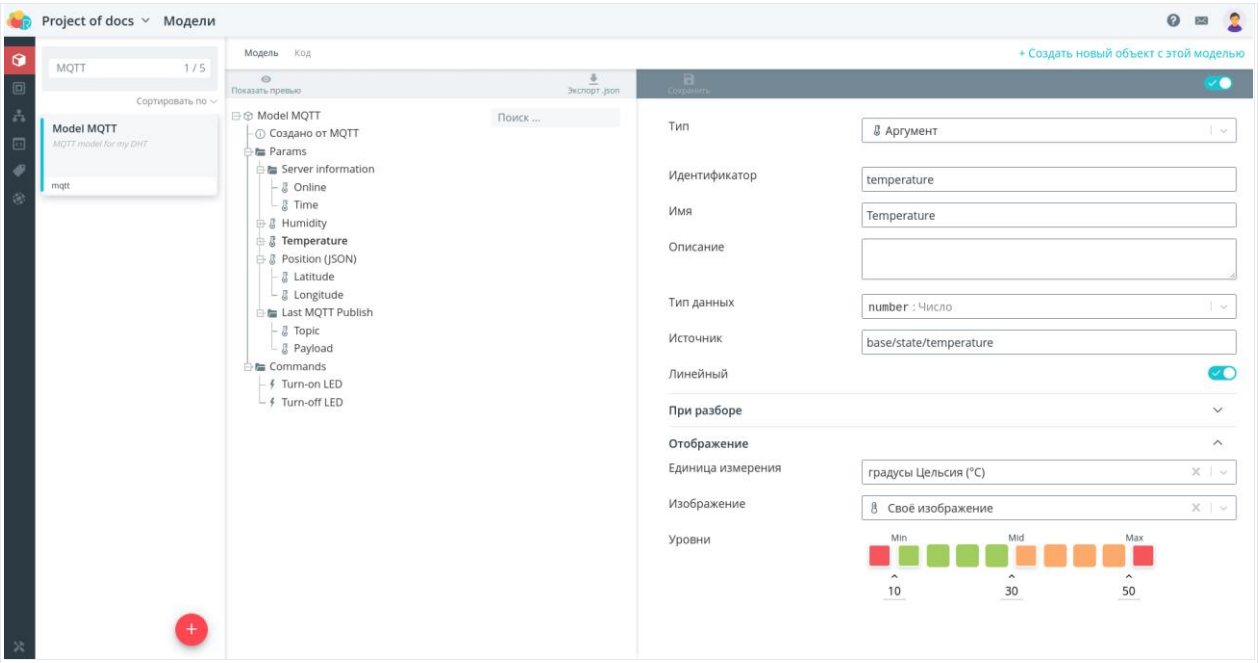


Рисунок 3. Редактирование модели

Параметры узлов модели

Сначала в каждом узле указывается его **тип**. В зависимости от выбранного типа формируются соответствующие поля формы для последующего заполнения.

Типы:

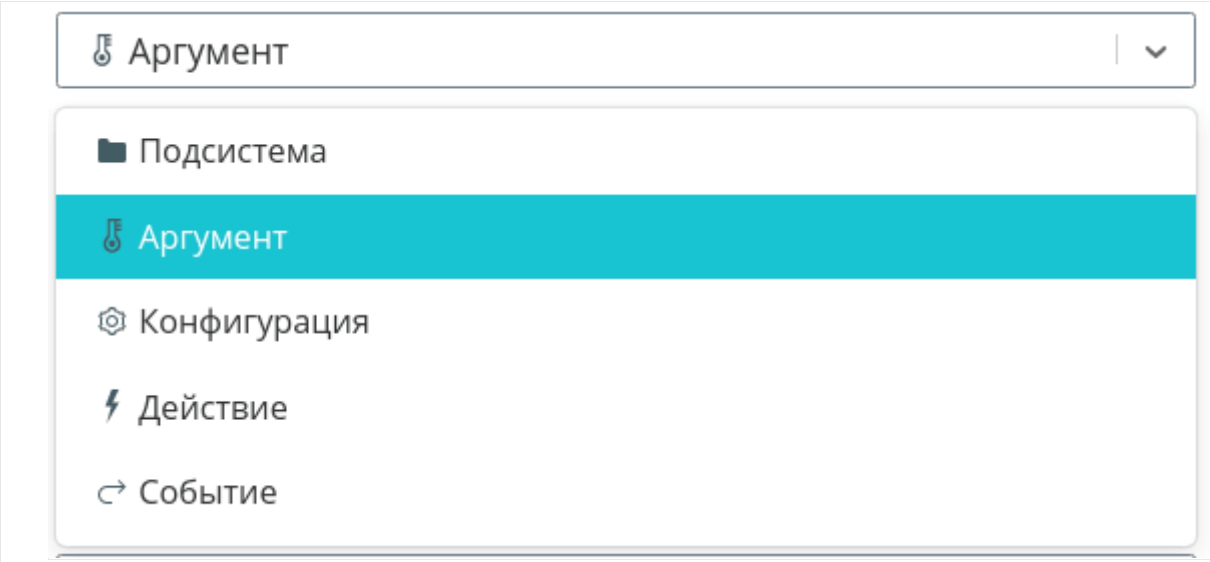
– **Подсистема** - тип узла, который служит для организации структуры модели, позволяя объединять параметры в группы. Подсистема подразумевает, что данный узел содержит в себе несколько элементов в виде ответвлений древовидной структуры.

– **Аргумент** – это параметр, который передает устройство на платформу (например, текущее измерение сенсора). Аргументы могут быть числовыми, логическими, строковыми или представлять собой объект или массив. При этом для числовых аргументов можно указать единицу измерения, изображение и задать уровни.

– **Конфигурация** – это параметр, значение которого затем можно задать в интерфейсе объекта. Его можно использовать в работе автоматов и обработчиков. По своей сути, узлы с данным типом — это константы, которые необходимы для хранения дополнительной информации об объекте. Это может быть, к примеру, максимально допустимая температура, объем топливного бака, длительность работы и т.п.

– **Действие** – это операция, которая нужна для отправки команды на устройство или запуска автомата.

– **Событие** – это наступление определенных условий, которое либо было зафиксировано объектом, либо произошло во внешних по отношению к объекту контроля приложениях. События используются в автоматах при построении переходов между состояниями. Именно по произошедшим с объектом событиям эти переходы и происходят.



У каждого типа есть пиктограмма, которая отображается слева от узла в дереве модели. Это позволяет визуально определять, к какому типу относится тот или иной узел.

Далее вводится уникальный по модели **идентификатор** и **имя** узла. Идентификатор необходим для распознавания узла в системе. Он является названием поля из данных о состоянии устройства при обращении к нему через API. Имя, в отличие от идентификатора, может быть неуникальным. Как идентификатор, так и имя генерируются при создании узла, их следует заменить на те идентификатор и имя, которые вам понятны и удобны.

Идентификатор должен быть уникальным! Идентификатор должен быть осмысленным! Идентификатор - для внутренней работы системы, имя - для интерфейса пользователя.

В платформе также заложены дополнительные функции для аргументов со специальными идентификаторами.

Идентификатор параметра	**Функция**
online	Статус подключения устройства: онлайн, подключено внешнее устройство онлайн, подключен бот платформы офлайн
_ts	Время получения пакета данных от устройства (хранится в микросекундах UTC). Если в модели не используется параметр с идентификатором time, то его функции выполняет _ts
Параметры, которые создает пользователь	
time	Время, по которому сортируются пакеты данных от устройства в истории, может отличаться от времени получения (_ts). Часто используется, когда устройство параллельно с актуальной информацией высылает исторические данные
lat, lon	Координаты перемещения объекта на карте

Идентификатор параметра	Функция
angle	Угол поворота иконки объекта на карте
x, y, z	Координаты перемещения объекта на схеме
speed	Скорость перемещения объекта на карте при использовании движения по маршруту для бота

Описание дает более полное представление о параметре. Оно выводится в качестве всплывающей подсказки при наведении на параметр в интерфейсе объекта.

Для узлов с типами “Подсистема” и “Событие” других полей заполнять не нужно. Описание полей для узлов с другими типами представлено ниже.

Для типа “Аргумент”

Для аргумента заполняются следующие поля:

- **тип данных**: выбирается один из возможных типов данных: числовой, логический, строковый, объект или массив. Тип данных служит для корректного отображения значений параметров в интерфейсе. Также для определенных типов есть дополнительные возможности:
  - числовой: указание множителя, единицы измерения, изображения и уровней перехода (разделы **При разборе** и **Отображение**);
  - логический: указание изображения и уровней перехода (разделы **При разборе** и **Отображение**);
  - объект: разбор данных в формате JSON (пример).
- **источник**: по нему происходит поиск значения параметра в структуре данных, полученных от устройства; строго зависит от протокола. Конфигурирование источников для конкретных протоколов подробно рассмотрено в разделе примеров;
- **линейный**: данный переключатель определяет, можно ли построить по параметру линейный график (плавно возрастающий или плавно убывающий): например, по температуре можно, а по широте или долготе - нет, такой график по отдельной координате не имеет смысла. При активации этого флага становится доступным добавление изображения в качестве иконки данного параметра и указание уровней индикации текущего значения параметра.

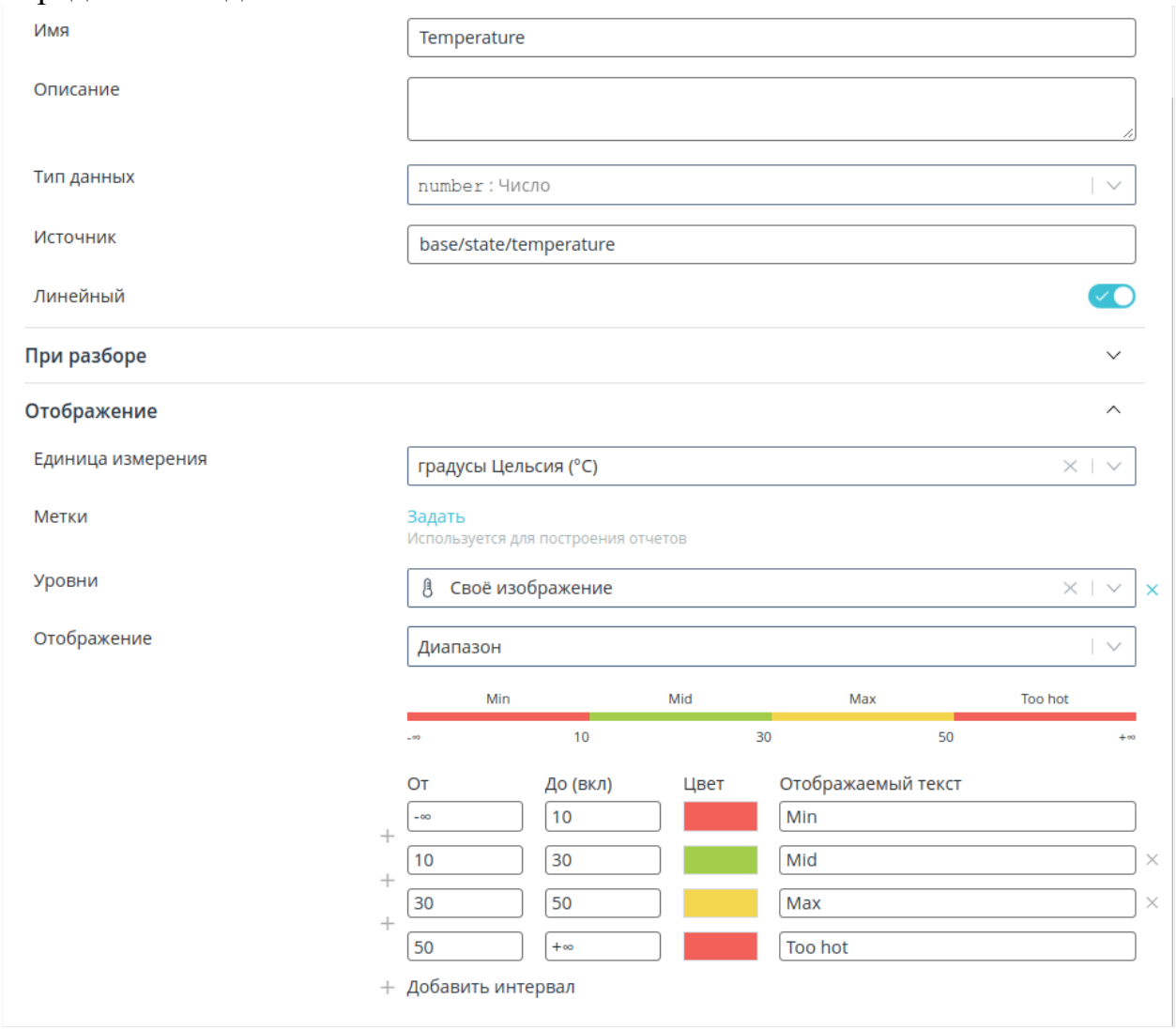
Разделы **При разборе** и **Отображение** открывают дополнительные поля:

- **множитель**: параметр, полученный от устройства, умножится на указанное значение (функция доступна только для чисел);
- **сохранять binary как**: служит для отображения данных в интерфейсе объекта в разном формате (например, Text - “ok”, Base64 - “b2s=”, HEX - “6F 6B”);
- **единица измерения**: для отображения рядом с числом; дополнительная функция есть у следующих единиц измерения:
  - проценты (%) - число помещается в шкалу, заполняющуюся согласно полученному значению процента;

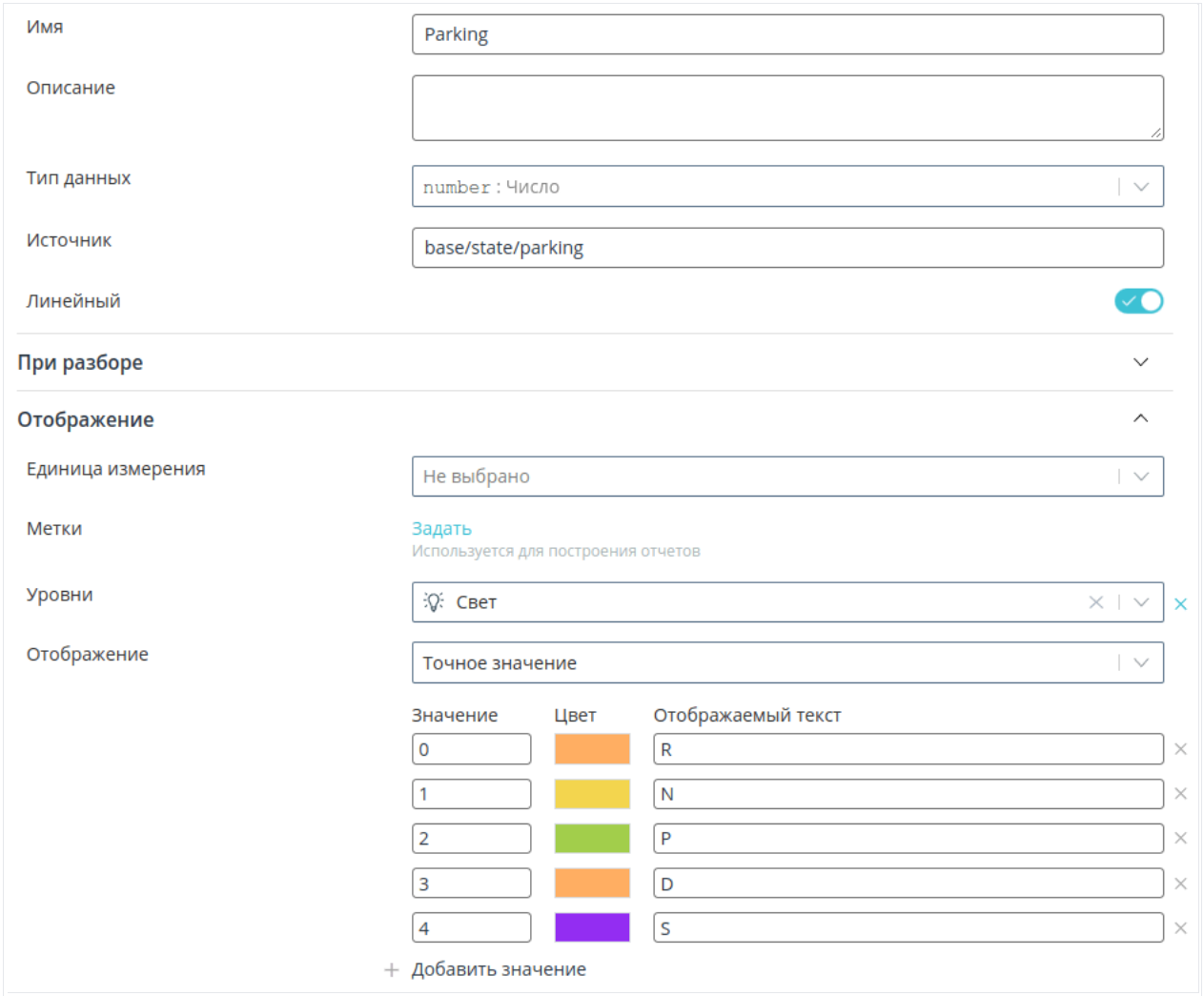
- миллисекунды (мс) - число, полученное в миллисекундах UNIX, конвертируется в удобочитаемый формат даты и времени.
- **изображение**: svg иконка для дополнительной индикации, представляет параметр в графическом виде на карточке объекта;
- **уровни**: пороги изменения цвета иконки и текста.

Настройка уровней отличается для аргументов по типам данных:

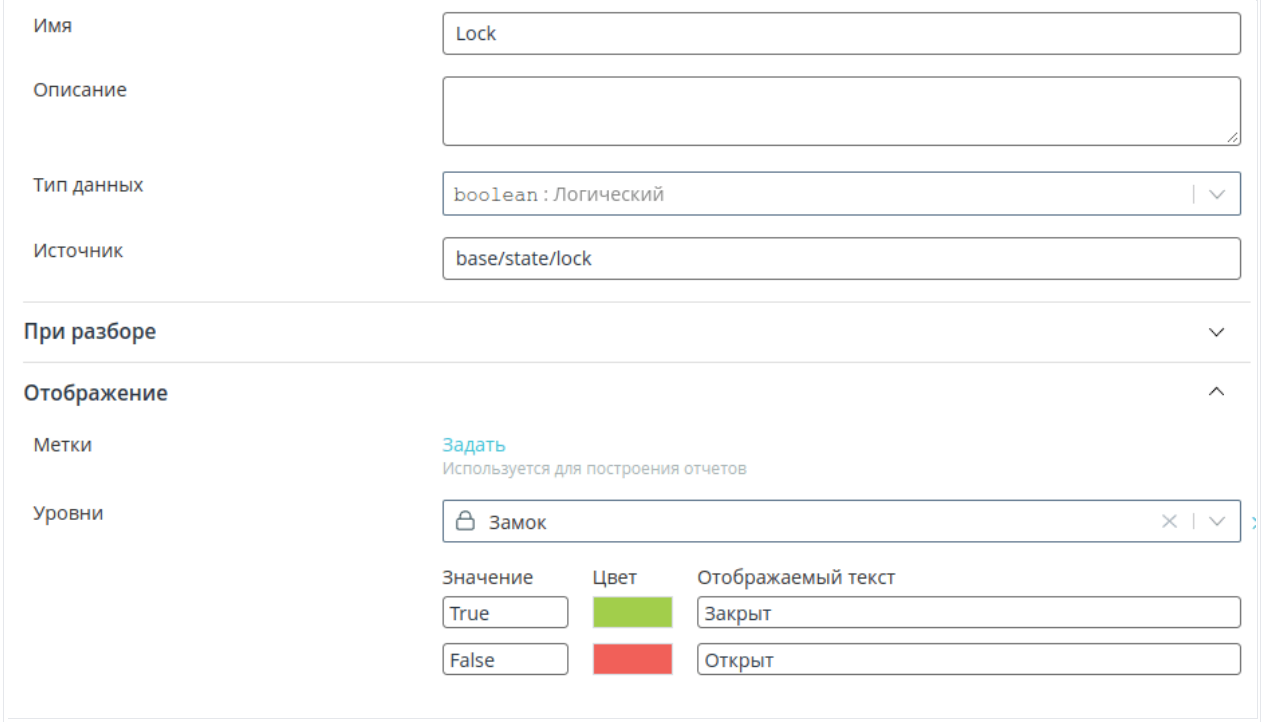
- **число**: изменение уровня при попадании полученной величины в определенный диапазон значений



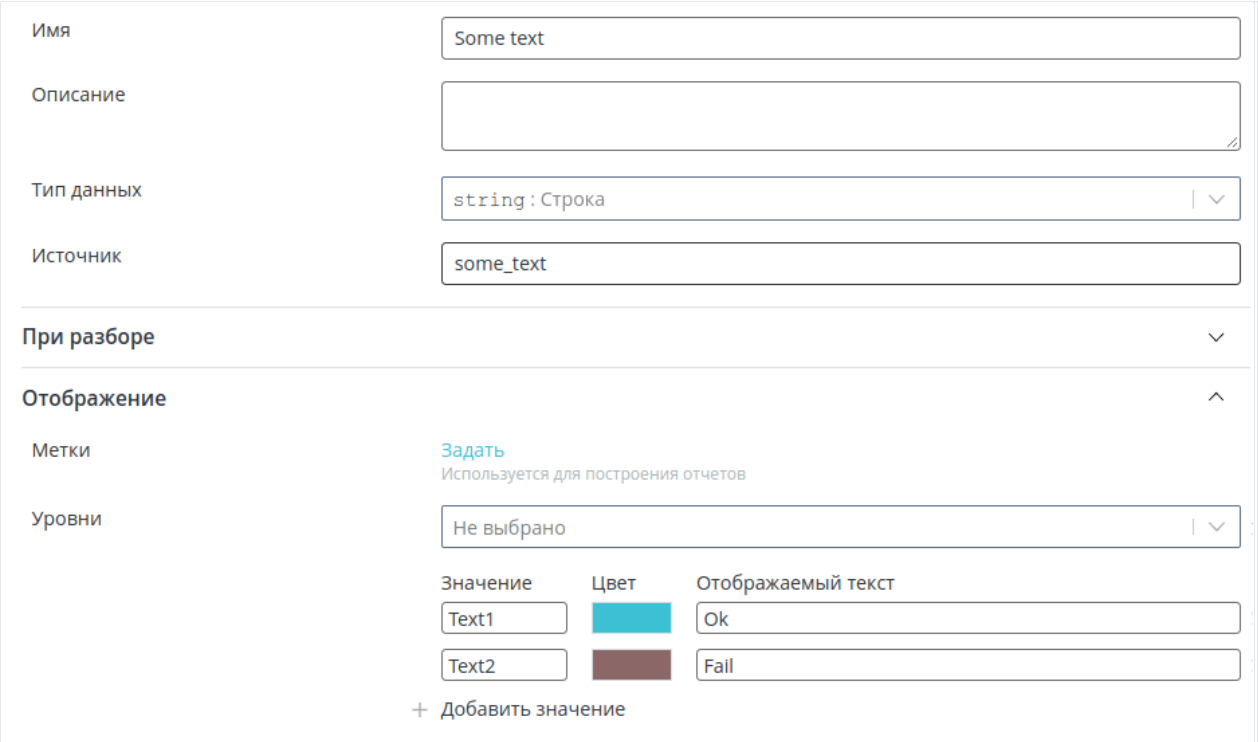
- изменение уровня при точном совпадении полученной величины с указанным значением



- **логический тип**



- **строка**



Иногда после загрузки собственного SVG изображения его цвет не меняется в соответствии с выставленными уровнями. В этом случае откройте картинку через любой текстовый редактор и уберите поле “fill”. После этого загрузите обновленное SVG изображение еще раз. Теперь его цвет будет меняться согласно заданным уровням.

Для типа “Конфигурация”

Для конфигурации заполняется только поле **Тип данных**.

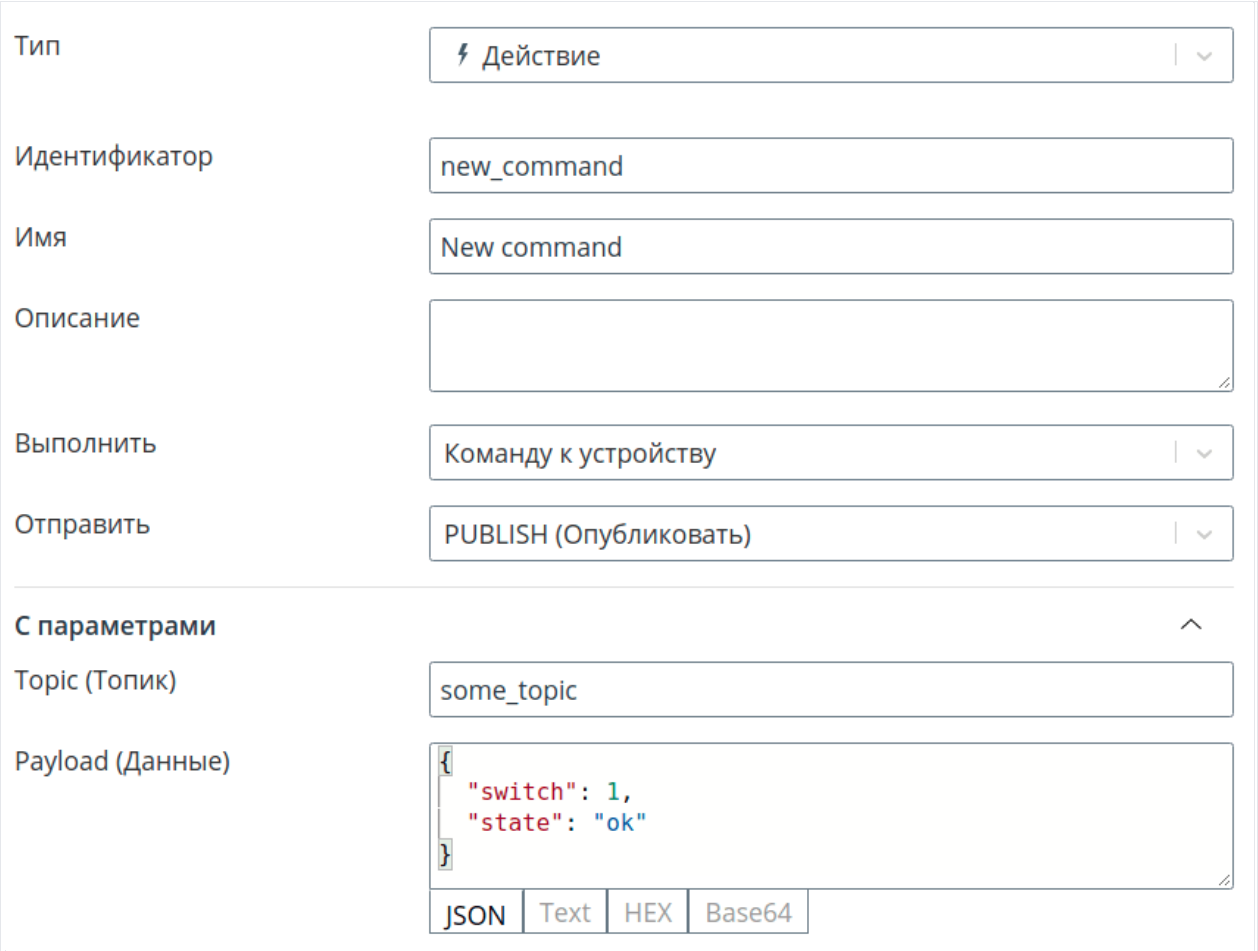
Для типа “Действие”

Для действия заполняется поле **выполнить**: выберите тип действия, который необходимо выполнять:

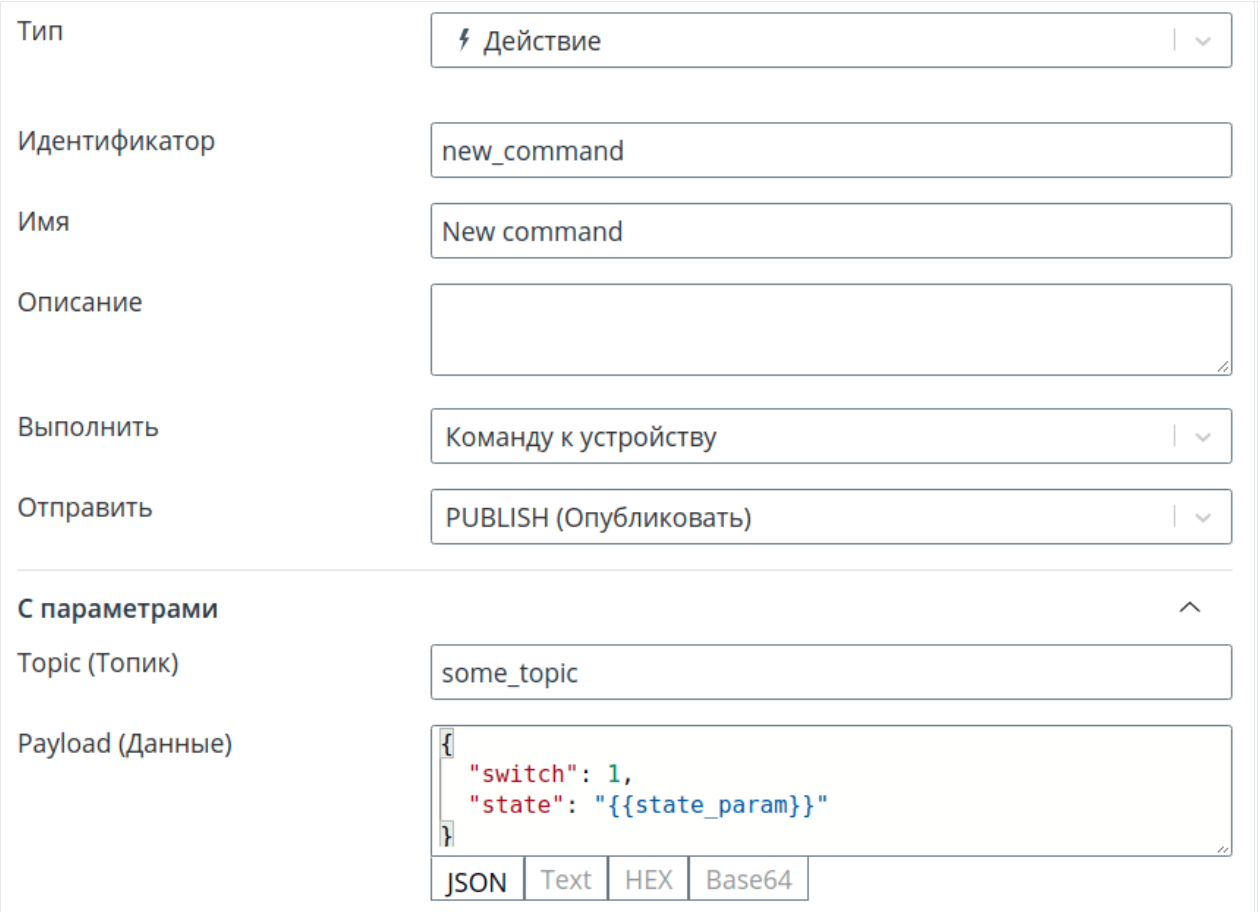
- **автомат**: в поле **Автомат** выберите тот автомат, который нужно запустить по команде;
- **команду к устройству**: в поле **Отправить** выберите тип команды в зависимости от протокола, после чего в разделе **С параметрами** укажите параметры команды.

В параметрах команды можно указать:

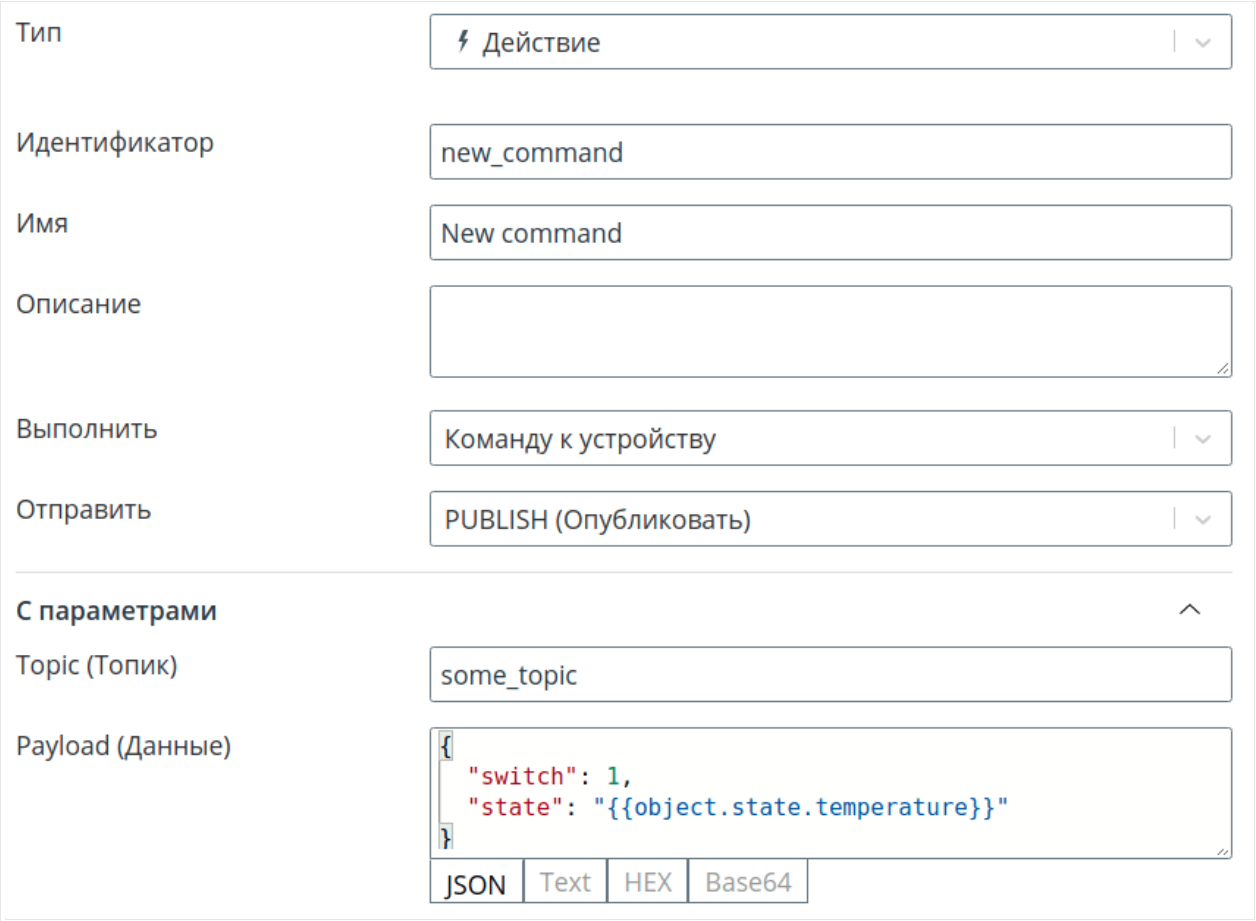
1. конкретные значения, которые будут каждый раз отправляться в команде;



2. параметры, значение которых можно указать непосредственно при отправке команды. Это нужно, если пользователь не знает заранее те значения, которые необходимо передавать в команде, либо эти значения раз от раза могут меняться. Для создания такого параметра необходимо при создании команды в модели объявить его в фигурных скобках {{param_name}};

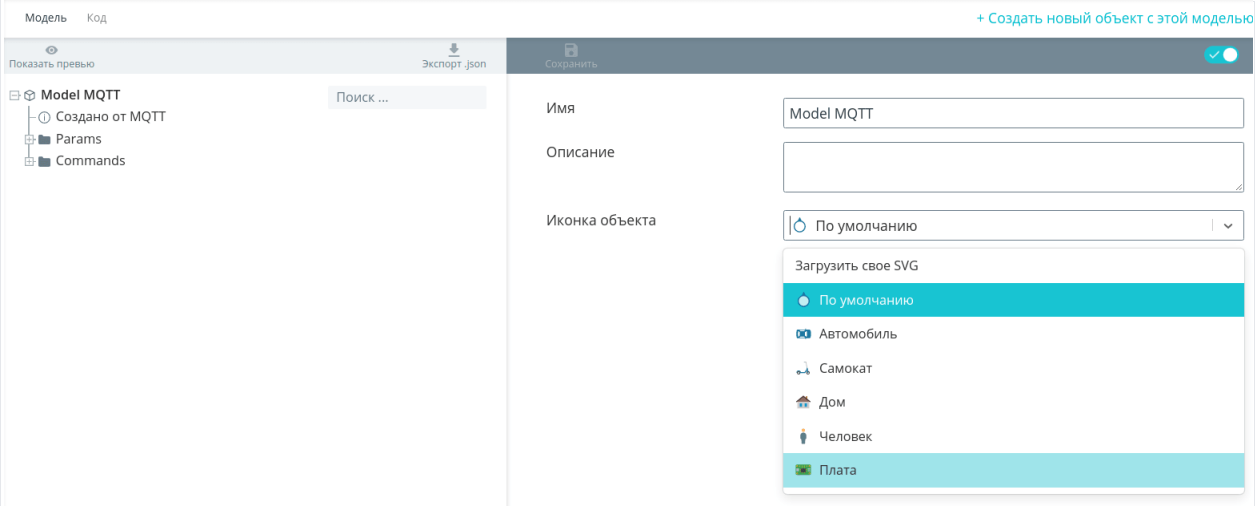


3. текущие параметры устройства. Для создания такого параметра необходимо при создании команды в модели указать в фигурных скобках object.state и после точки указать идентификатор параметра из модели (например, {{object.state.temperature}}). При отправке команды подставится значение этого параметра из журнала.



Для корневого узла

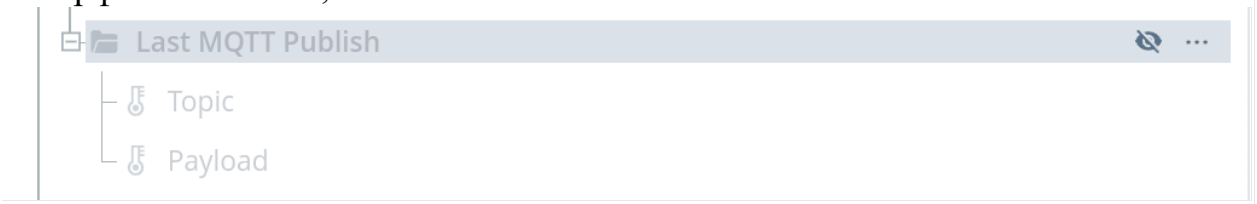
Для корневого узла модели помимо имени и описания можно задать иконку. Все объекты с такой моделью отобразятся на карте с заданной иконкой. При этом можно использовать как стандартные иконки, так и загрузить собственное изображение в формате SVG.



Формирование структуры модели

Справа от каждого узла модели при наведении появляются иконка “глаз” и три точки, при нажатии на которые открываются функции по работе с узлами.

• **Скрыть (иконка “глаз”)** - скрытие элемента из интерфейса объекта. Этот параметр не будет обрабатываться платформой и отображаться в журнале, но при необходимости его можно вернуть. В случае скрытия узла с типом **Подсистема** параметры, которые в нее входят, также скрываются в интерфейса объекта;



• **Добавить** - создание нового узла в виде ответвления для выбранного элемента на уровень ниже;

• **Копировать** - копирование выбранного элемента. При этом копируется вся заполненная по узлу информация с тем отличием, что у идентификатора приписывается сгенерированный постфикс, а у имени добавляется надпись ”(copy)”;

• **Удалить** - удаление выбранного элемента.

Все элементы по модели перемещаются с помощью drag-and-drop. Так вы определяете структуру модели для вашего устройства удобным образом.

Все параметры и команды необходимо разносить по смысловым группам - подсистемам. Это позволит организовать удобный интерфейс отображения в объекте.

Для того чтобы посмотреть, как будет выглядеть в журнале состояние объекта и интерфейс отправки команд, нажмите **Показать превью**.

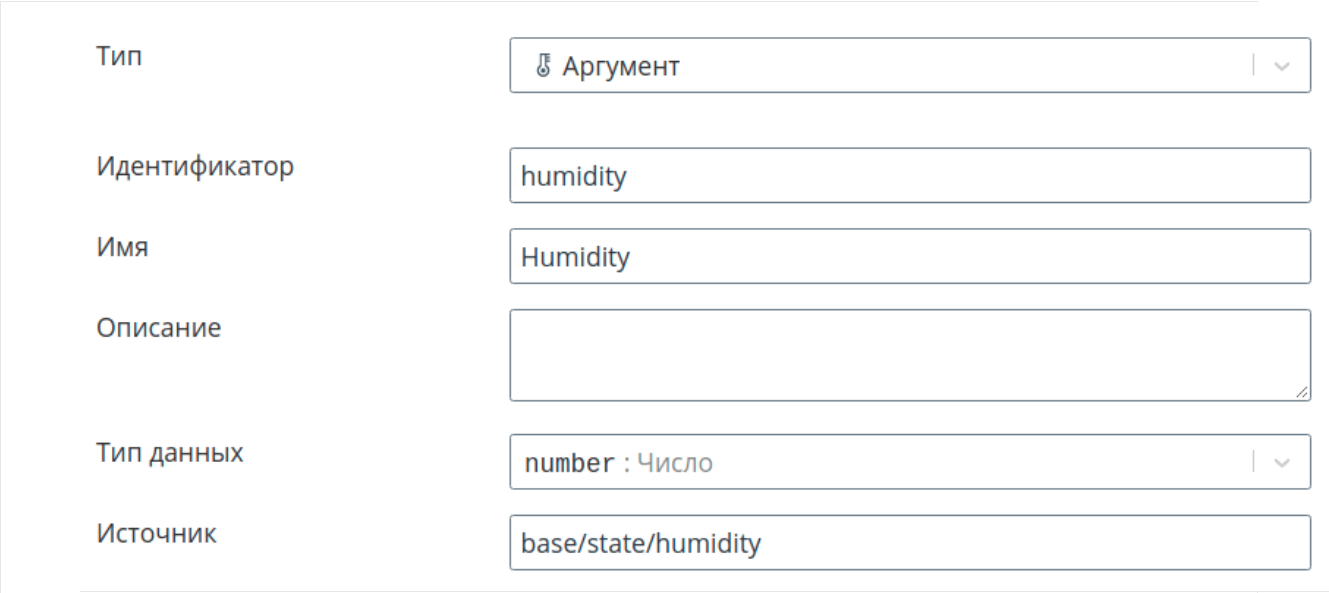
Также любой элемент модели можно редактировать и просматривать на вкладке **Код**. Полная структура модели представлена там в формате JSON.

MQTT

Аргументы

В случае MQTT источник данных для аргумента - это топик, по которому устройство отправляет значение этого параметра.

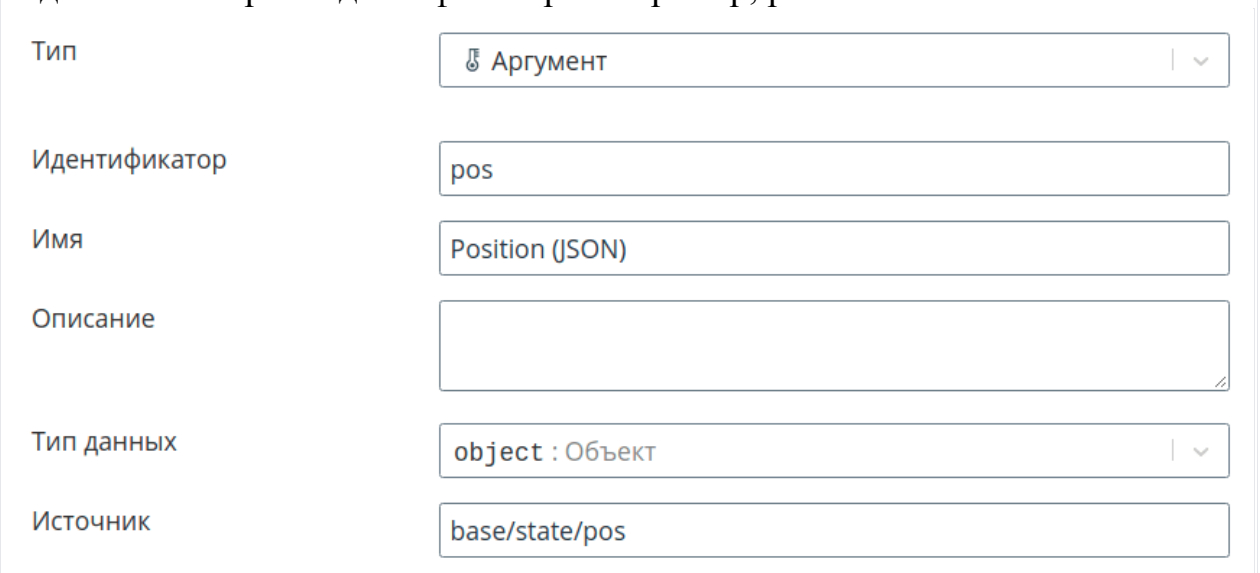
Например, если устройство отправляет значение влажности в пакете с топиком base/state/humidity, то именно этот топик необходимо указать в качестве источника в данном параметре.



Для того чтобы отправить несколько параметров в одном пакете, воспользуйтесь форматом JSON при построении payload. Например, отправить две координаты в одном пакете можно, указав топик как base/state/pos, а payload в виде { "lat": 55.55, "lon": 33.33 }

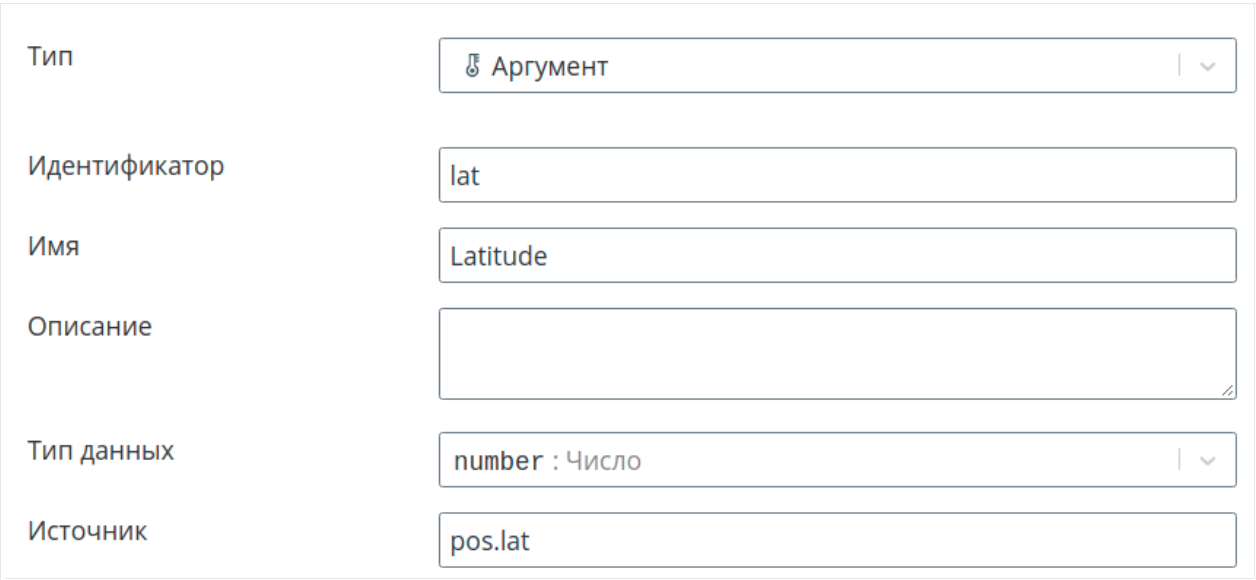
Как же разобрать такой сложный пакет по модели? На помощь приходит параметр с типом данных **Объект**:

1. Создайте аргумент с типом данных **Объект**, в источнике укажите тот топик, по которому ожидаете данные - в данном случае base/state/pos. Задайте некоторый идентификатор - например, pos.

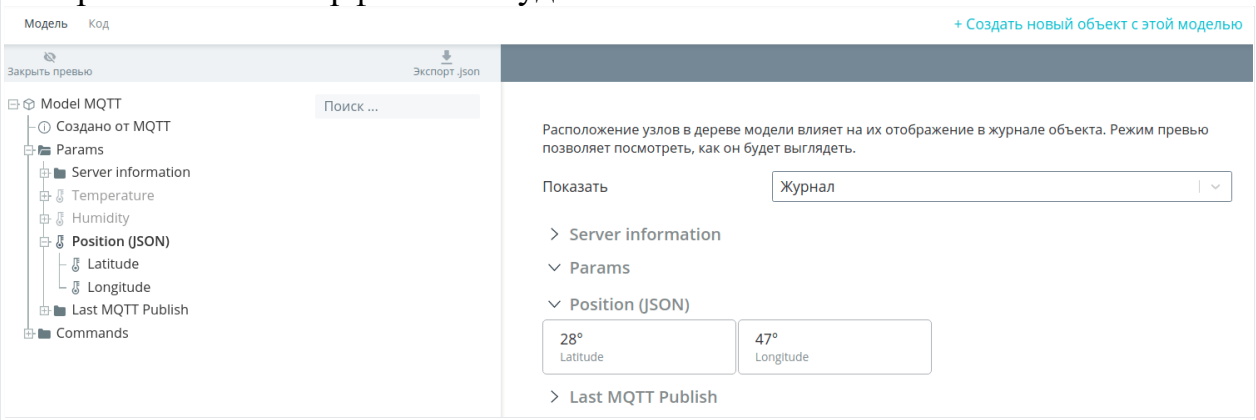


2. Создайте еще два узла - под каждое поле из JSON. В данном случае это узлы для широты и долготы. У каждого из этих параметров укажите тип данных, соответствующий типу этих значений в JSON. В нашем пакете координаты передаются как числа, поэтому укажите тип данных **Число**.

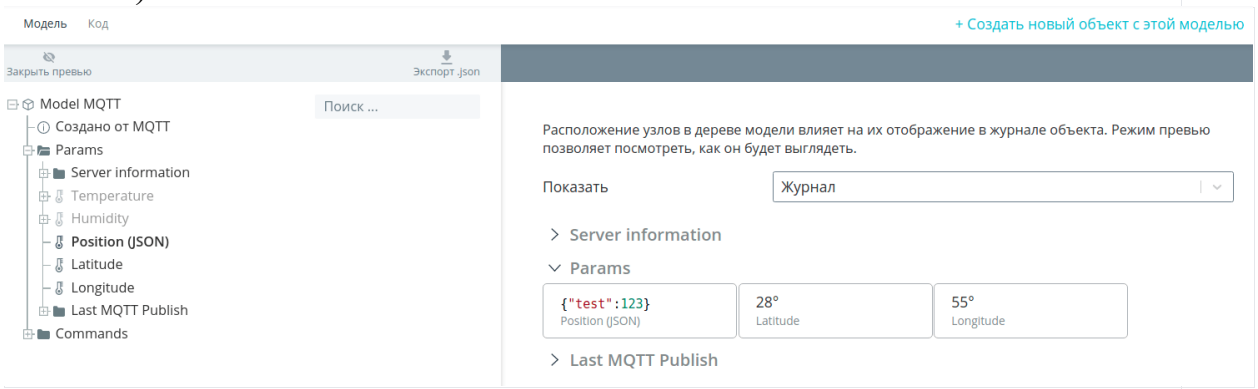
3. Источник у каждого из узлов сформируйте в виде <идентификатор_узла_с_типом_объект>.<нужное_поле_из_json>. Для координат из примера источники получаются следующие: pos.lat и pos.lon.



Если аргументы для полей из JSON вы укажете на уровень ниже аргумента-объекта, тогда аргумент-объект выступит как подсистема и отображаться в интерфейсе не будет.



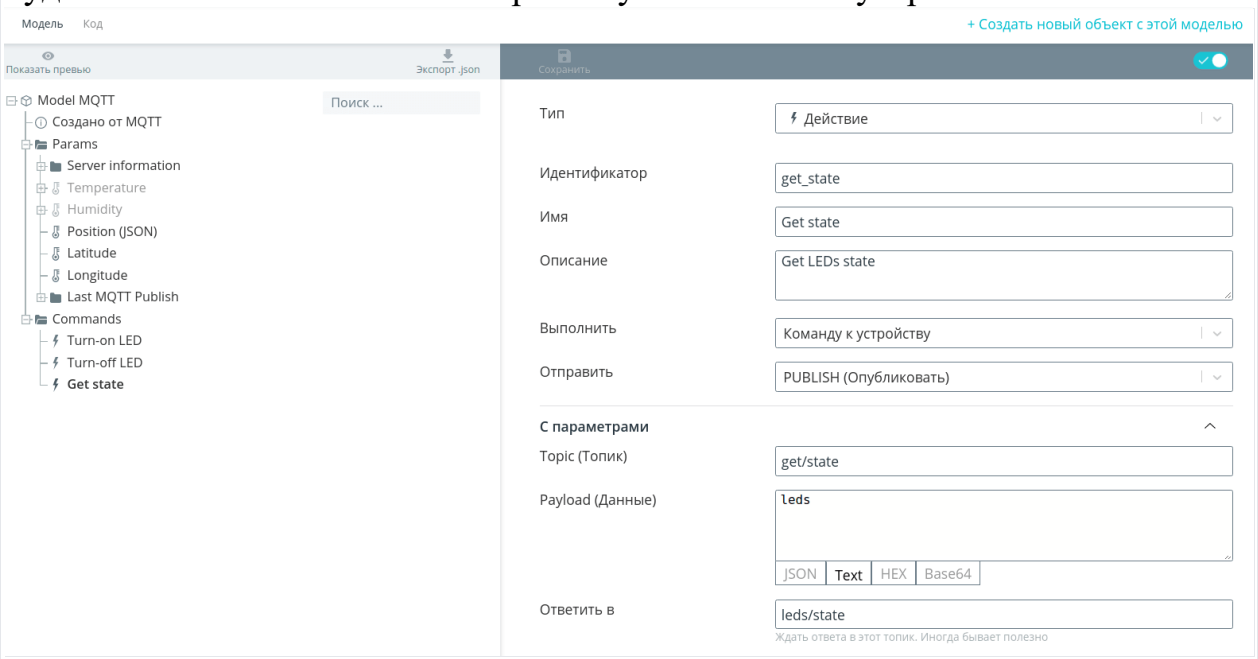
Если же аргументы для полей из JSON находятся на одном уровне с аргументом-объектом, тогда в интерфейсе объекта отобразятся как аргументы для полей (в данном случае в виде чисел), так и аргумент для объекта (в виде строки JSON).



Действия

В случае MQTT при заполнении действия в поле **Отправить** выберите **PUBLISH (Опубликовать)**. После этого укажите, с какими параметрами необходимо опубликовать сообщение:

- **Topic (Топик)**, с которым передается команда в виде сообщения на устройство;
- **Payload (Данные)** - полезная нагрузка, содержащая текст передаваемого сообщения;
- **Ответить в** - топик, при получении пакета с которым команда считается выполненной. Как вариант, можно создать команду для получения состояния устройства и указать в поле **Ответить в** топик, по которому устройство отправляет свое состояние - например, leds/state. Тогда при отправке этой команды, даже если вы получите от устройства Ack (PUBACK – Publish acknowledgement) или какие-то пакеты с другими топиками, команда не будет считаться выполненной до тех пор, пока не придет пакет именно с указанным нами топиком leds/state. Если поле **Ответить в** не заполнено, то команда будет считаться выполненной при получении Ack от устройства.



Задание по подключению MQTT устройства к облачной платформе Rightech

Цель: подключить и настроить устройство с поддержкой MQTT на платформе Rightech IoT Cloud, используя внешний MQTT-брокер.

1. Регистрация на платформе Rightech IoT Cloud:
Перейдите на [Rightech IoT Cloud](https://rightech.io) и зарегистрируйтесь.
После регистрации зайдите в свою учетную запись.

2. Создание модели устройства:
В личном кабинете перейдите в раздел «Модели».
Нажмите «Создать модель» и выберите шаблон для MQTT устройства.
Настройте параметры модели:
Название модели.
Параметры данных (тип данных, параметры передачи).
Добавьте свойства (например, температура, влажность) или другие необходимые параметры для мониторинга устройства.

3. Настройка MQTT брокера:
Если MQTT брокер ещё не установлен, скачайте и установите MQTT брокер Mosquitto с [официального сайта](https://mosquitto.org/download/).
После установки настройте брокер, указав следующие параметры:
Порт (по умолчанию 1883 или 8883 для защищенного соединения).
Разрешите доступ для устройства через публичную сеть или используйте локальный сервер для тестов.
Проверьте соединение с брокером, чтобы убедиться в корректной работе.

4. Добавление устройства в облако Rightech:
- В разделе «Объекты» на платформе Rightech добавьте новое устройство.
- Выберите ранее созданную модель и укажите параметры подключения:
  - Адрес MQTT брокера.
  - Порт (1883 или другой, который вы настроили).
  - Топики для подписки и публикации данных (например, `device/data`).
  - Логин и пароль для безопасного подключения, если это требуется вашим брокером.

5. Настройка физического устройства:
Программируйте устройство для подключения к MQTT брокеру.
Укажите:
IP адрес брокера.
Порт подключения.
Топики для отправки данных.
Установите параметры передачи данных, такие как интервал отправки.

Запустите устройство и убедитесь, что оно подключено к брокеру и публикует данные.

6. Мониторинг данных на платформе:

Перейдите в панель управления в Rightech IoT Cloud.

Проверьте, поступают ли данные от устройства в реальном времени.

Настройте графики, фильтры, уведомления или триггеры, чтобы мониторить изменения показателей устройства.

7. Тестирование и отладка:

В случае отсутствия данных проверьте:

Соединение устройства с брокером.

Корректность топиков MQTT.

Логи на платформе и устройстве.

Убедитесь, что данные передаются корректно и отображаются в интерфейсе платформы.

Пример MQTT запросов:

Например, с mosquitto_pub клиентом из проекта Eclipse Mosquitto.

```
$ mosquitto_pub -d -h dev.rightech.io -i <ric-mqtt-client-id> -t
base/state/temperature -m 23
```

Публикация данных с устройства:

Топик: device/data

```
mosquitto_pub -h broker_address -t device/data -m '{"temperature": 25}'
```

Подписка на команды:

Топик: device/commands

```
mosquitto_sub -h broker_address -t device/commands
```

Задание практической работы №9

Часть 1. Регистрация на платформе Rightech IoT Cloud

Rightech IoT Cloud — это бескодовая (no-code) IoT-платформа для быстрого создания прикладных проектов интернета вещей. Для регистрации на платформе необходимо перейти по данной ссылке https://dev.rightech.io/

Программный продукт Rightech IoT Cloud (RIC, рус. Райтек ИоТ Клауд) от компании-разработчика КОМНЭТ является фреймворк-инструментом и предназначен для быстрого создания разработчиками приложений интернета вещей) Платформа RIC реализована на принципах универсализации и имеет низкую зависимость от конкретного оборудования и протоколов, что позволяет легко объединять разнородные устройства в едином разрабатываемом на платформе решении.

Часть 2. Создание виртуальных устройств в облаке

Согласно варианту по номеру бригады создайте в облаке виртуальные устройства для получения данных. Каждое устройство должно иметь свой профиль, соответствующий передаваемым на устройство данным. В качестве протокола для профилей устройств используйте MQTT.

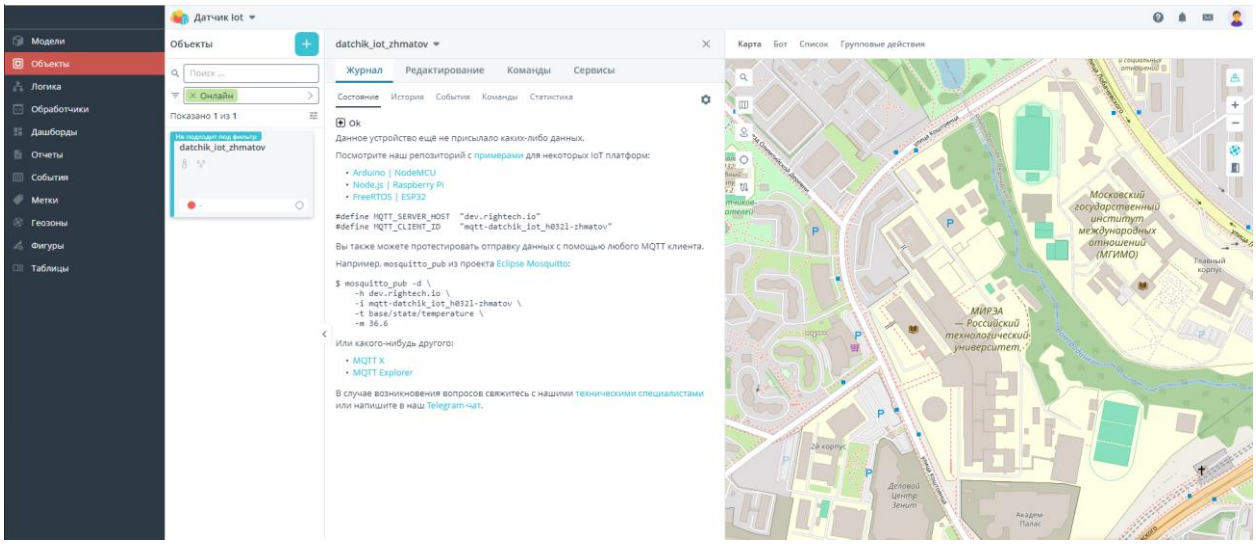
№ варианта	Датчики
1	1. Датчик температуры 2. Датчик движения 3. Датчик напряжения
2	1. Датчик шума 2. Датчик освещенности 3. Датчик напряжения
3	1. Датчик шума 2. Датчик качества воздуха 3. Датчик напряжения
4	1. Датчик движения 2. Датчик температуры 3. Датчик напряжения
5	1. Датчик качества воздуха 2. Датчик освещенности 3. Датчик напряжения
6	4. Датчик влажности 5. Датчик шума 6. Датчик напряжения
7	7. Датчик влажности 8. Датчик температуры 9. Датчик напряжения

Часть 3. Отправка данных в облако

Выполните передачу тестовых данных в каждое из созданных устройств. Данные должны соответствовать типу устройства, то есть, к примеру, в термометр должна

поступить температура. Данные можно передавать при помощи утилиты mosquito_pub. Ссылка на документацию по передаче данных на устройства по MQTT.

В отчете необходимо отразить созданные устройства, процесс отправки данных с облако, а также отображение этих данных на виртуальных устройствах в облачной платформе.



Дополнительное задание практической работы №9

Часть 1. Ознакомьтесь с существующими облачными решениями Интернета вещей, выберите одно из этих решений о обоснуйте свой выбор применения именного этого варианта в реализуемом проекте. Предлагаемые платформа: Rightech IoT Cloud.

Часть 2. Реализуйте отправку данных с физического устройства (программного эмулятора) в выбранную облачную платформу по выбранному протоколу. В отчете необходимо представить листинг кода подключения к облаку, а также обоснование выбора протокола.

Литература и ресурсы:

1. Rightech IoT Cloud: [Документация](https://dev.rightech.io)
2. MQTT протокол: [Описание](https://mqtt.org)
3. Mosquitto MQTT: [Инструкция по установке](https://mosquitto.org/man/mosquitto-8.html)