

Методические указания по выполнению практических работ № 9-12

«Облачные платформы интернета вещей»

по дисциплине «Технологические основы Интернета вещей»

Оглавление

Практическая работа №9 – Знакомство с облачными платформами IoT	2
Определение облака и его назначение	2
Облака в концепции Интернета вещей.....	2
Основные технологии для облачных платформ интернета вещей	4
Подключение конечных устройств к платформам Интернета вещей	5
Задание практической работы №9	9
Дополнительное задание практической работы №9	10
Практическая работа №10 – Управление устройствами при помощи платформ Интернета вещей	11
Облачная логика	11
Механизм правил платформы ThingsBoard	11
Пример создания цепочки правил	14
Задание практической работы №10.....	20
Дополнительное задание практической работы №10	21
Практическая работа №11 – Реакции платформ Интернета вещей на входящие данные	22
Назначение реакций	22
Механизм тревог платформы ThingsBoard	22
Пример работы с тревогами	23
Задание практической работы №11	29
Дополнительное задание практической работы №11	29
Практическая работа №12 – Отправка оповещения от облачной платформы	31
Использование сторонних сервисов для отправки оповещений	31
Способы оповещений на платформе ThingsBoard	31
Пример работы Email оповещениями.....	32
Задание практической работы №12	36
Дополнительное задание практической работы №12	36
Требования к отчету по ПР №9-12:	37
Литература для изучения:	37

Практическая работа №9 – Знакомство с облачными платформами IoT

Определение облака и его назначение

Под облаком понимается сервер или группа серверов, доступ к которым осуществляется удаленно при помощи сети Интернет. Облака применяются для хранения и обработки (облачные вычисления) передаваемых в них данных.

Облачное хранилище данных (англ. cloud storage) — модель онлайн-хранилища, в котором данные хранятся на многочисленных распределённых в сети серверах, предоставляемых в пользование клиентам, в основном, третьей стороной. В отличие от модели хранения данных на собственных выделенных серверах, приобретаемых или арендуемых специально для подобных целей, количество или какая-либо внутренняя структура серверов клиенту, в общем случае, не видна. Данные хранятся и обрабатываются в так называемом «облаке», которое представляет собой, с точки зрения клиента, один большой виртуальный сервер. Физически же такие серверы могут располагаться удалённо друг от друга географически.

Облачные вычисления (англ. cloud computing) — модель обеспечения удобного сетевого доступа по требованию к некоторому общему фонду конфигурируемых вычислительных ресурсов (например, сетям передачи данных, серверам, устройствам хранения данных, приложениям и сервисам — как вместе, так и по отдельности), которые могут быть оперативно предоставлены и освобождены с минимальными эксплуатационными затратами или обращениями к провайдеру.

Облака в концепции Интернета вещей

Концепция Интернета вещей использует облачные ресурсы как хранения, так и для обработки всех данных. Однако этим функционал облаков IoT не ограничивается. Здесь появляется другое понятие IoT-платформа.

IoT-платформа – это программное обеспечение системы интернета вещей для подключения конечных устройств (датчиков, сенсоров, контроллеров и т.д.) к облаку и удаленного доступа к ним. Целью IoT-платформы является обеспечение бесшовной интеграции различных аппаратных средств с помощью специальных интерфейсов, протоколов связи, сетевых топологий, а также средств хранения, обработки и интеллектуального анализа данных.

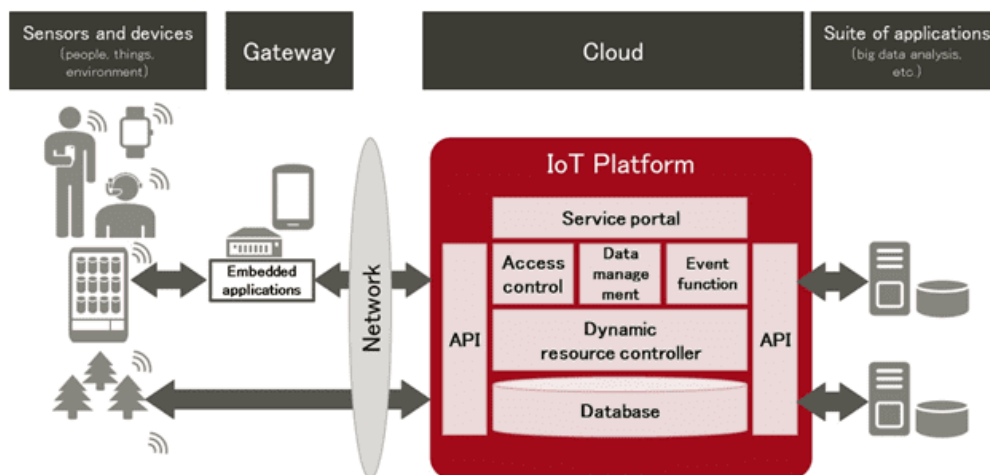


Рисунок 1 – Архитектура IoT решения

Выделяют следующие 8 обязательных компонентов полноценной IoT-платформы:

- **связь и нормализация (Connectivity & normalization)** – сведение различных протоколов и форматов данных в один программный интерфейс, гарантирующий точную передачу информации и взаимодействие со всеми конечными устройствами.
- **управление устройствами (Device management)** – обеспечение корректной и бесперебойной работы конечных устройств, их конфигурирование и обновление программных приложений на них и пограничных шлюзах.
- **база данных (Database)** – обеспечение масштабируемого и надежного хранения информации.
- **обработка и управление действиями (Processing & action management)** – мониторинг текущего и прогнозирование будущего состояния технологического оборудования на основании данных с конечных устройств, которые на нем установлены, а также выработка команд для изменения состояний оборудования и передача этих сигналов на исполнительное устройство;
- **аналитика (Analytics)** – интеграция и кластеризация данных, а также прогнозирование значений искомых параметров, в т.ч. с использованием методов машинного обучения (Machine Learning);
- **визуализация (Visualization)** – наглядное представление собранной, обработанной и проанализированной информации в виде графиков, диаграмм, таблиц и других понятных представлений;
- **дополнительные инструменты (Additional tools)** – средства, которые позволяют разработчикам ПО и DevOps-инженерам расширить функциональные возможности платформы Internet of Things с помощью графического интерфейса и программирования;
- **внешние интерфейсы (External interfaces)** – API, SDK и шлюзы для интеграции с другими сервисами, системами и платформами.

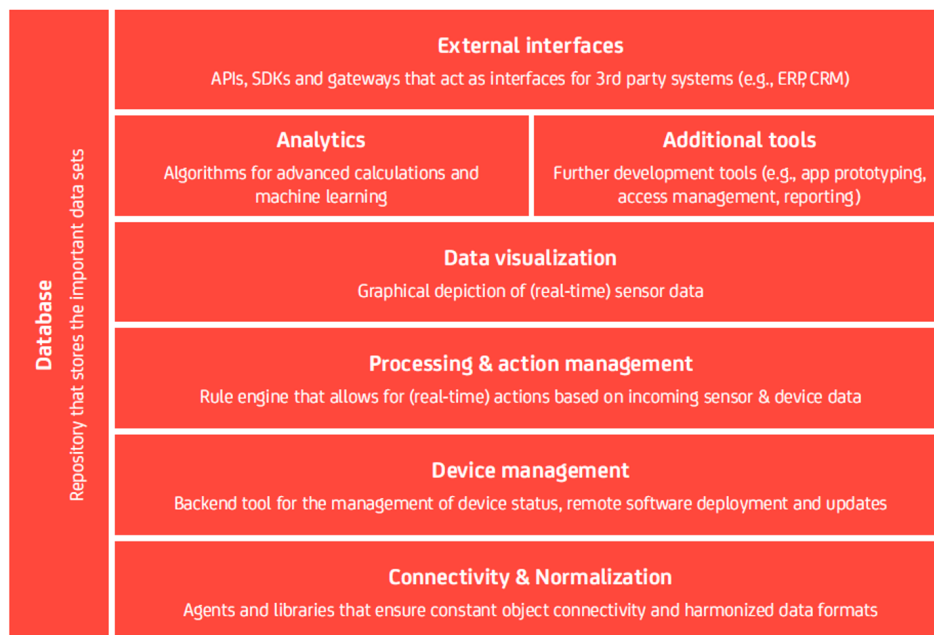


Рисунок 2 – Обязательные компоненты IoT-платформы

Основные технологии для облачных платформ интернета вещей

Эталонная архитектура платформ интернета вещей описана международным стандартом «ISO/IEC 30141:2018. Internet of Things – Reference Architecture». Этот стандарт, пока не переведенный полностью на русский язык, необходим для обеспечения единого фреймворка DevOps-инженеров, разработчиков IoT-платформ и приложений интернета вещей с целью создания надежных, безопасных и устойчивых к сбоям решений. Разработка IoT-продуктов в рамках эталонной архитектуры позволяет бизнесу усилить эффект от внедрения новых технологий, снизив риски от их использования.

Большинство современных IoT-платформ обеспечивают интеллектуальный анализ информации в реальном времени с использованием следующих инструментов Big Data:

- агрегирование и фильтрация потоков данных (Storm, Samza);
- поддержка пакетных операций с накопленным набором Big Data (средствами Hadoop, Spark);
- предиктивная аналитика с использованием методов Machine Learning потоковых и пакетных данных (Spark, MLlib).

Также IoT-платформы используют следующие технологии Big Data:

- прикладные протоколы семейства TCP/IP — CoAP, HTTP/HTTPS;
- протоколы обмена сообщениями в концепции «издатель-подписчик» (MQTT, AMQP, XMPP, DDS), реализованные в программных брокерах RabbitMQ, Apache Qpid, Apache ActiveMQ, а также Apache Kafka, который считается наиболее масштабируемым инструментом управления очередью;
- средства быстрой загрузки потоковых данных со шлюза и конечных устройств (Apache NiFi, Apache MiNiFi, Apache Flume).

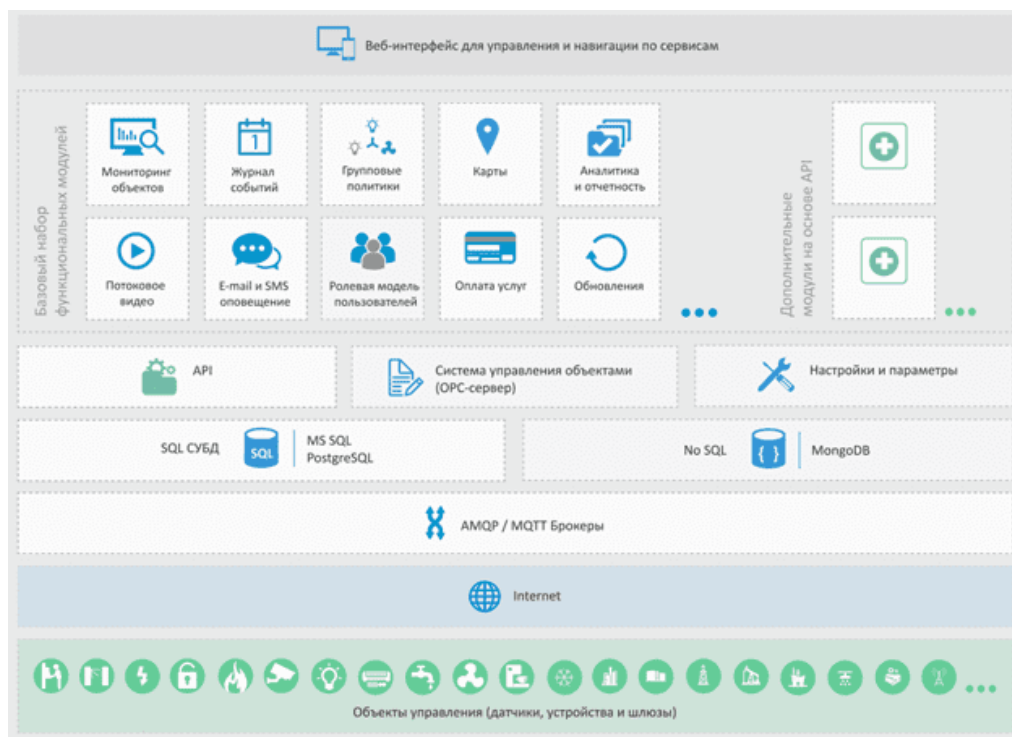


Рисунок 3 – Типовая архитектура IoT-платформы

Подключение конечных устройств к платформам Интернета вещей

В качестве примера IoT-платформы будем использовать open source платформу под названием Things Board (<https://thingsboard.io/>). Данная платформа включает все требуемые компоненты IoT-платформы, позволяющие реализовывать различные сервисы Интернета вещей.

Создание виртуального устройства

Первым этапом работы с любой IoT-платформой является подключение физических устройств, которые будут передавать необходимые данные. В облаке создаются виртуальные копии реальных устройств, которые служат точками подключения к облаку, передающими данные во внутренние механизмы облака. На платформе ThingsBoard такие устройства можно найти на вкладке Devices. (Рисунок 3)

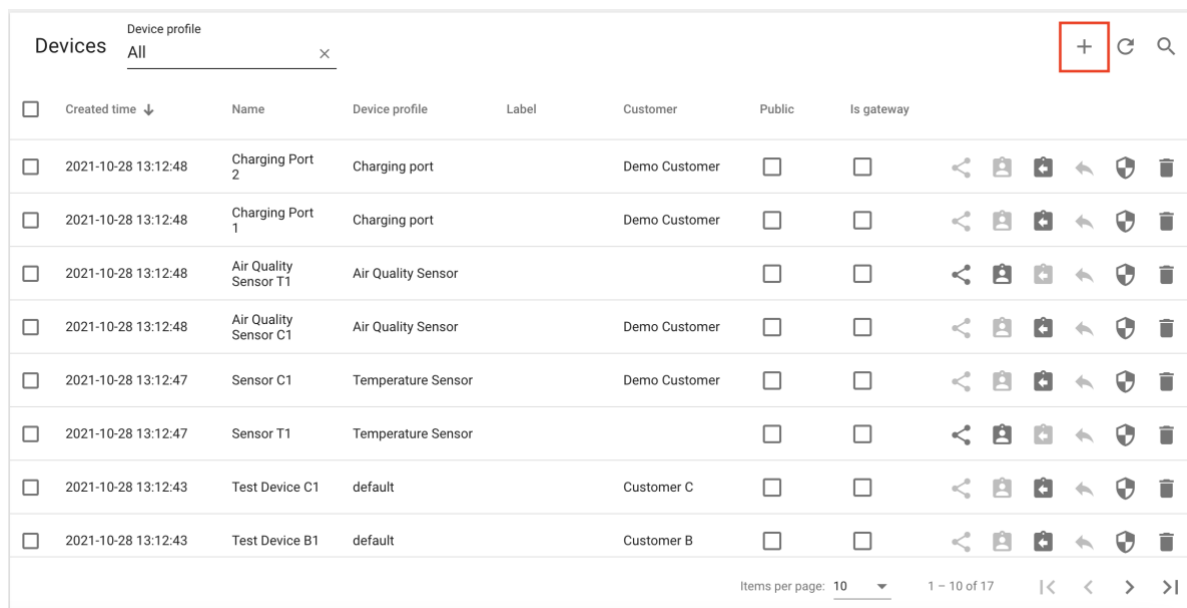
The screenshot shows the 'Devices' page in the ThingsBoard interface. The table below represents the data visible in the 'Devices' list.

Created time	Name	Device profile	Label	Customer	Public	Is gateway	Actions
2021-10-28 13:12:48	Charging Port 2	Charging port		Demo Customer	<input type="checkbox"/>	<input type="checkbox"/>	[Icons]
2021-10-28 13:12:48	Charging Port 1	Charging port		Demo Customer	<input type="checkbox"/>	<input type="checkbox"/>	[Icons]
2021-10-28 13:12:48	Air Quality Sensor T1	Air Quality Sensor		Demo Customer	<input type="checkbox"/>	<input type="checkbox"/>	[Icons]
2021-10-28 13:12:48	Air Quality Sensor C1	Air Quality Sensor		Demo Customer	<input type="checkbox"/>	<input type="checkbox"/>	[Icons]
2021-10-28 13:12:47	Sensor C1	Temperature Sensor		Demo Customer	<input type="checkbox"/>	<input type="checkbox"/>	[Icons]
2021-10-28 13:12:47	Sensor T1	Temperature Sensor		Demo Customer	<input type="checkbox"/>	<input type="checkbox"/>	[Icons]
2021-10-28 13:12:43	Test Device C1	default		Customer C	<input type="checkbox"/>	<input type="checkbox"/>	[Icons]
2021-10-28 13:12:43	Test Device B1	default		Customer B	<input type="checkbox"/>	<input type="checkbox"/>	[Icons]

Items per page: 10 | 1 - 10 of 17

Рисунок 4 – Вкладка Devices

В качестве примера создадим виртуальный датчик влажности, получающий данные по протоколу MQTT. Вызвать окно создания устройства можно по кнопке в правой верхней части списка устройств (рисунок 6).



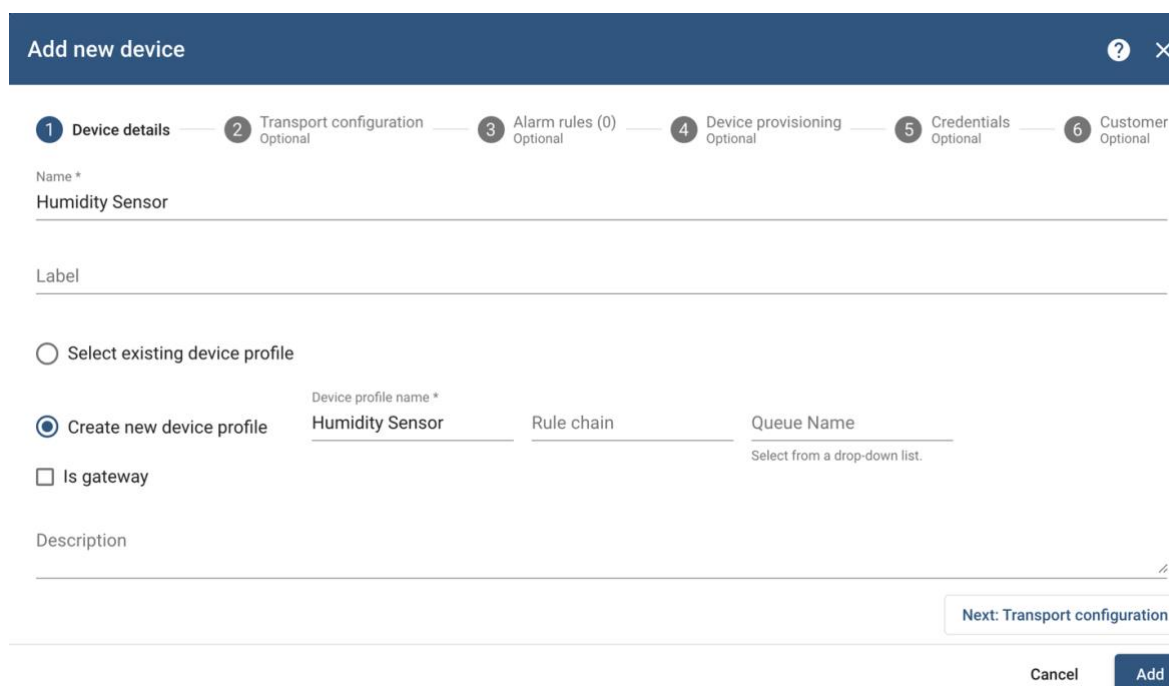
Devices								Device profile	
								All	×
<input type="checkbox"/>	Created time ↓	Name	Device profile	Label	Customer	Public	Is gateway		
<input type="checkbox"/>	2021-10-28 13:12:48	Charging Port 2	Charging port		Demo Customer	<input type="checkbox"/>	<input type="checkbox"/>		
<input type="checkbox"/>	2021-10-28 13:12:48	Charging Port 1	Charging port		Demo Customer	<input type="checkbox"/>	<input type="checkbox"/>		
<input type="checkbox"/>	2021-10-28 13:12:48	Air Quality Sensor T1	Air Quality Sensor			<input type="checkbox"/>	<input type="checkbox"/>		
<input type="checkbox"/>	2021-10-28 13:12:48	Air Quality Sensor C1	Air Quality Sensor		Demo Customer	<input type="checkbox"/>	<input type="checkbox"/>		
<input type="checkbox"/>	2021-10-28 13:12:47	Sensor C1	Temperature Sensor		Demo Customer	<input type="checkbox"/>	<input type="checkbox"/>		
<input type="checkbox"/>	2021-10-28 13:12:47	Sensor T1	Temperature Sensor			<input type="checkbox"/>	<input type="checkbox"/>		
<input type="checkbox"/>	2021-10-28 13:12:43	Test Device C1	default		Customer C	<input type="checkbox"/>	<input type="checkbox"/>		
<input type="checkbox"/>	2021-10-28 13:12:43	Test Device B1	default		Customer B	<input type="checkbox"/>	<input type="checkbox"/>		

Items per page: 10 1 – 10 of 17

Рисунок 5 – Кнопка создания устройства

Необходимо создать именно новое устройство, а не импортировать его откуда-то.

Окно создания содержит в себе несколько важных элементов, которые необходимо задать. Первый – это наименование устройства (name), оно необходимо для его идентификации пользователем. Второй важный параметр – это профиль устройства (device profile). Профиль позволяет объединять устройства в группы, что может облегчить управление устройствами и помочь при обработке поступающих данных. Управлять самими профилями можно на вкладке Device Profiles. Создадим устройство с наименованием Humidity Sensor, а также создадим для него новый профиль с таким же наименованием (Рисунок 7).



Add new device
?
×

1 Device details
2 Transport configuration Optional
3 Alarm rules (0) Optional
4 Device provisioning Optional
5 Credentials Optional
6 Customer Optional

Name *
Humidity Sensor

Label

☐ Select existing device profile

☒ Create new device profile

Device profile name *
Humidity Sensor

Rule chain

Queue Name
Select from a drop-down list.

☐ Is gateway

Description

Next: Transport configuration

Cancel Add

Рисунок 6 – Создание устройства

Параметры Rule chain, а также Queue Name пока нас не интересуют, они будут рассмотрены в дальнейшем.

Теперь необходимо настроить протокол передачи данных на создаваемое устройство. В целом IoT-платформы поддерживают широкий перечень протоколов, как клиент-серверного типа, так и протоколы очередей сообщений. К примеру, рассматриваемая платформа ThingsBoard позволяет подключаться по следующим протоколам: MQTT, CoAP, LWM2M и HTTP. Поскольку прошлые практики рассматривались на примере протокола MQTT, продолжим работу с ним и настроим создаваемое устройство на этот протокол. Для этого перейдем на вкладку Transport Configuration.

Если задать в качестве типа Default, с устройством можно будет общаться по любому из доступных протоколов. Зададим возможность работы только по MQTT (Рисунок 8).

Также на данной вкладке можно выбрать формат передачи данных в облако. Оставим в качестве формата JSON.

Все последующие вкладки можно пропустить, на текущей стадии они не понадобятся.

The screenshot shows the 'Add new device' interface with the 'Transport configuration' step active. The transport type is set to 'MQTT'. Under 'MQTT device topic filters', the telemetry filter is 'v1/devices/me/telemetry' and the attributes filter is 'v1/devices/me/attributes'. The MQTT device payload is set to 'JSON'. The interface includes a progress bar at the top with steps: Device details, Transport configuration (current), Alarm rules (0), Device provisioning, Credentials, and Customer. At the bottom, there are 'Back', 'Next: Alarm rules', 'Cancel', and 'Add' buttons.

Рисунок 7 – Параметры подключения к устройству

Передача данных на виртуальное устройство

Платформа ThingsBoard имеет унифицированный MQTT API для передачи данных на виртуальные устройства. Все данные по умолчанию необходимо передавать в топик `v1/devices/me/telemetry`. Идентификация получающего устройства осуществляется по уникальному токenu, который присуждается каждому создаваемому устройству. Найти этот токен можно в окне устройства, открывающемся при клике на нем в списке устройств (Рисунок 9).

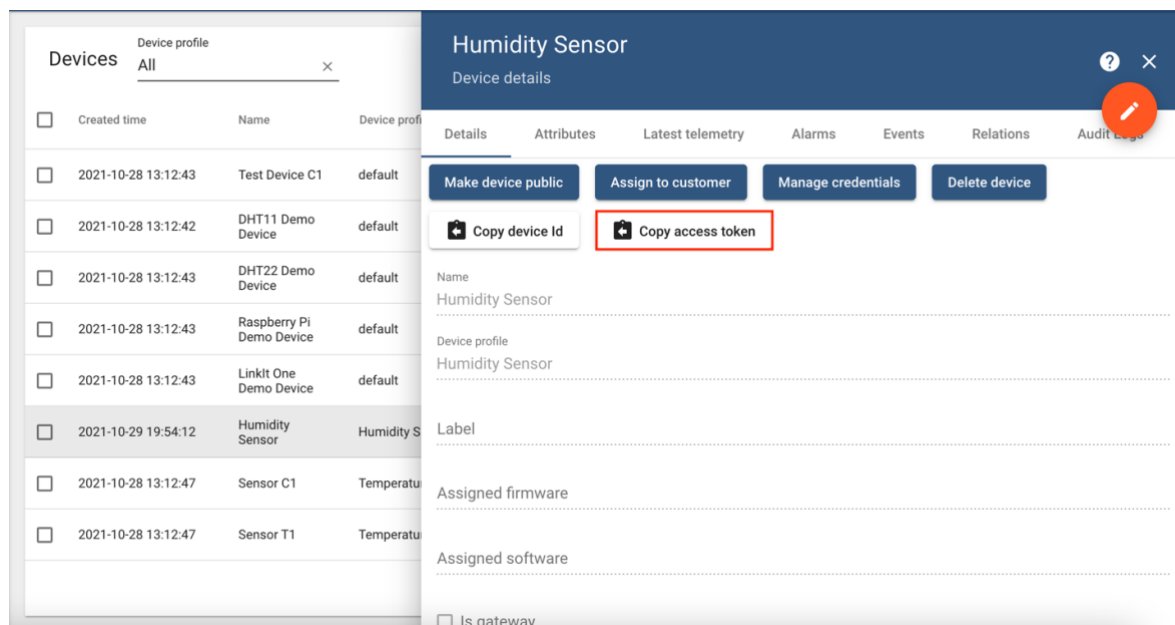


Рисунок 8 – Окно характеристик устройства

Попробуем отправить данные на созданное устройство при помощи утилиты `mosquitto_pub`. Для этого можно обратиться к документации по MQTT API (<https://thingsboard.io/docs/reference/mqtt-api/>). Из документации следует, что можно отправить сообщение, задав следующие параметры для `mosquitto_pub`:

```
mosquitto_pub -d -q 1 -h "demo.thingsboard.io" -t "v1/devices/me/telemetry" -u "<access_token>" -m "<json_string>"
```

Попробуем отправить данные о влажности на только что созданное устройство.

```
Client (null) sending CONNECT
Client (null) received CONNACK (0)
Client (null) sending PUBLISH (d0, q1, r0, m1, 'v1/devices/me/telemetry', ... (13 bytes))
Client (null) received PUBACK (Mid: 1, RC:0)
Client (null) sending DISCONNECT
```

Рисунок 9 – Результат выполнения команды `mosquitto_pub`

Благодаря флагу `-d` можно отследить процесс установки соединения и передачи данных.

Теперь можно проверить поступление информации в облако. Для этого необходимо открыть информацию о созданном устройстве и перейти на вкладку Last telemetry (Рисунок 11). Если всё было выполнено правильно, там будет отображаться отправленная информация.

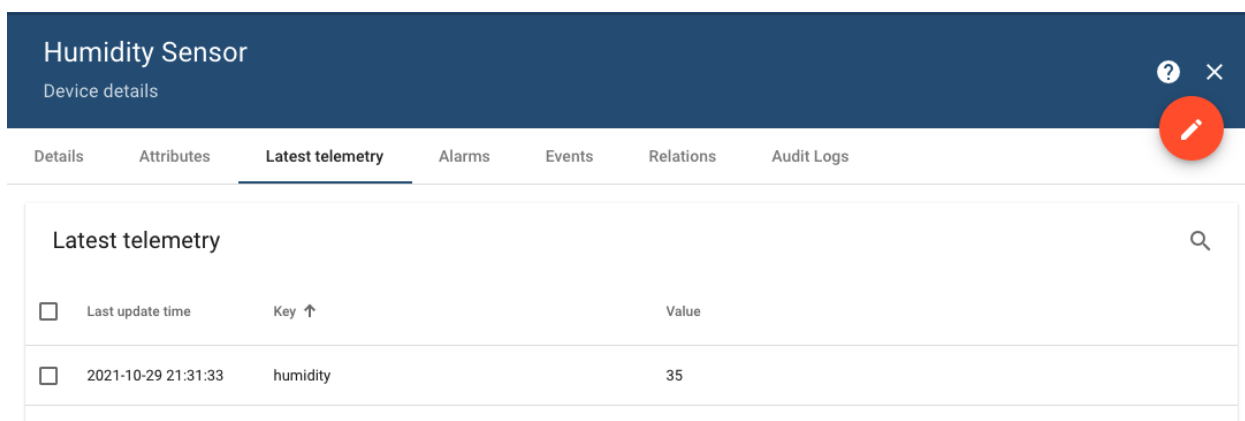


Рисунок 10 – Вкладка Last telemetry

Таким образом происходит передача информации с физических устройств в облачную IoT-платформу.

Задание практической работы №9

Часть 1. Регистрация на платформе ThingsBoard

ThingsBoard имеет тестовый сервер в сборке Community Edition для проверки доступных функций платформы и тестирования своих приложений. Для регистрации на платформе необходимо перейти по данной ссылке <https://demo.thingsboard.io/signup>.

Часть 2. Создание виртуальных устройств в облаке

Согласно варианту 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. Остальные параметрами оставьте незаполненными.

№ 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	1. Датчик влажности 2. Датчик шума 3. Датчик напряжения
7	1. Датчик влажности 2. Датчик температуры 3. 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 `mosquitto_pub`. Ссылка на документацию по передаче данных на устройства по MQTT – <https://thingsboard.io/docs/reference/mqtt-api/>.

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 и обоснуйте свой выбор применения именно этого варианта в реализуемом проекте. Предлагаемые платформы: Adafruit, ThingsBoard, RiiTech IoT Cloud, Azure IoT и т.д.

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.

Практическая работа №10 – Управление устройствами при помощи платформ Интернета вещей

Облачная логика

Облачные платформы берут на себя обработку логики работы IoT-решения. Перенос логики в облако позволяет сократить нагрузку на устройства Интернета вещей, которым остается лишь передавать данные и получать управляющие команды, зависящие от передаваемых данных. Также перемещение основных алгоритмов обработки в облако позволяет применять более серьезные технологии обработки данных, к примеру, машинное обучение, нейронные сети и т.д. не ограничивая их эффективность возможностями микроконтроллеров.

В первом блока практических работ необходимо было описать правила реакции стенда Wirenboard на изменения состояния его составляющих. Облачные платформы могут взять задачу обработки данных правил на себя. Рассмотрим на примере платформы ThingsBoard реализацию подобных облачных сценариев.

Механизм правил платформы ThingsBoard

Rule Engine позволяет реагировать на возникающие в облачной платформе события.

Основные концепции

Сообщение

Сообщение механизма правил – это сериализуемая неизменяемая структура данных, которая представляет различные сообщения в системе.

Например:

- Входящая телеметрия, обновление атрибутов или вызов RPC с устройства;
- Событие жизненного цикла объекта: создано, обновлено, удалено, назначено, не назначено, атрибуты обновлены;
- Событие состояния устройства: подключено, отключено, активно, неактивно и т. д.;
- Другие системные события.

Все примеры будут подробнее рассмотрены ниже.

Сообщение содержит следующую информацию:

- Идентификатор сообщения: универсальный уникальный идентификатор на основе времени;
- Отправитель сообщений: Устройство или идентификатор другой сущности;
- Тип сообщения: «Post telemetry» или «Inactivity Event» и т. д.;
- Полезная нагрузка сообщения: тело JSON с фактической полезной нагрузкой сообщения;
- Метаданные: список пар ключ-значение с дополнительными данными о сообщении.

Узел правила

Узел правил – это базовый компонент механизма правил, который обрабатывает одно входящее сообщение за раз и создает одно или несколько исходящих сообщений. Узел правил может фильтровать, обогащать, преобразовывать входящие сообщения, выполнять действия или взаимодействовать с внешними системами.

Связь узла правила

Узлы правил могут быть связаны с другими узлами правил. Каждое отношение имеет тип отношения, а также метку, используемую для определения логического значения отношения. При соединении выхода одного узла с входом другого узла всегда указывает тип отношения, который используется для маршрутизации сообщений.

Типичные отношения узла правила - «Успех» и «Неудача». Узлы правил, которые представляют логические операции, могут использовать «Истина» или «Ложь». Некоторые конкретные узлы могут использовать совершенно разные типы отношений, например: “Post Telemetry”, “Attributes Updated”, “Entity Created”, и т. д.

Цепочка правил

Цепочка правил – это логическая группа узлов правил и их отношений. Пример формирования цепочки правил будет рассмотрен ниже.

Результат обработки сообщения

Возможны три результата обработки сообщения: успех, сбой и тайм-аут. Попытка обработки сообщения помечается как «Успешная», когда последний узел правила в цепочке обработки успешно обработал сообщение. Попытка обработки сообщения помечается как «Сбой», если один из узлов правил выдает «Сбой» обработки сообщения, и нет узлов правила, которые могли бы обработать этот сбой. Попытка обработки сообщения помечается как «Тайм-аут», когда общее время обработки превышает настраиваемый порог.

Типы узлов правил

Все доступные узлы правил сгруппированы в соответствии с их характером:

- Узлы фильтрации используются для фильтрации и маршрутизации сообщений;
- Узлы обогащения используются для обновления метаданных входящего сообщения;
- Узлы преобразования используются для изменения полей входящего сообщения, таких как «Источник», «Тип», «Полезная нагрузка», «Метаданные»;
- Узлы действий выполняют различные действия на основе входящего сообщения;
- Внешние узлы используются для взаимодействия с внешними системами.

Более подробно с каждым узлом можно ознакомиться в документации к ThingsBoard.

Конфигурация

Каждый узел правил может иметь определенные параметры конфигурации, которые зависят от реализации узла правил. Например, узел «Filter-script» настраивается с помощью пользовательской JS-функции, обрабатывающей входящие данные. Конфигурация узла

«Внешний - отправка электронной почты» позволяет указать параметры подключения к почтовому серверу.

Окно конфигурации узла правил можно открыть, дважды щелкнув узел в редакторе цепочки правил:



Рисунок 11 – Окно конфигурации узла правил

Проверка функции JavaScript

Некоторые узлы правил имеют особую функцию пользовательского интерфейса, которая позволяет пользователям тестировать написанный JS-код. После того, как вы нажмете на кнопку **TEST FILTER FUNCTION**, вы увидите редактор JS, который позволяет вам заменять входные параметры и проверять выходные данные функции.

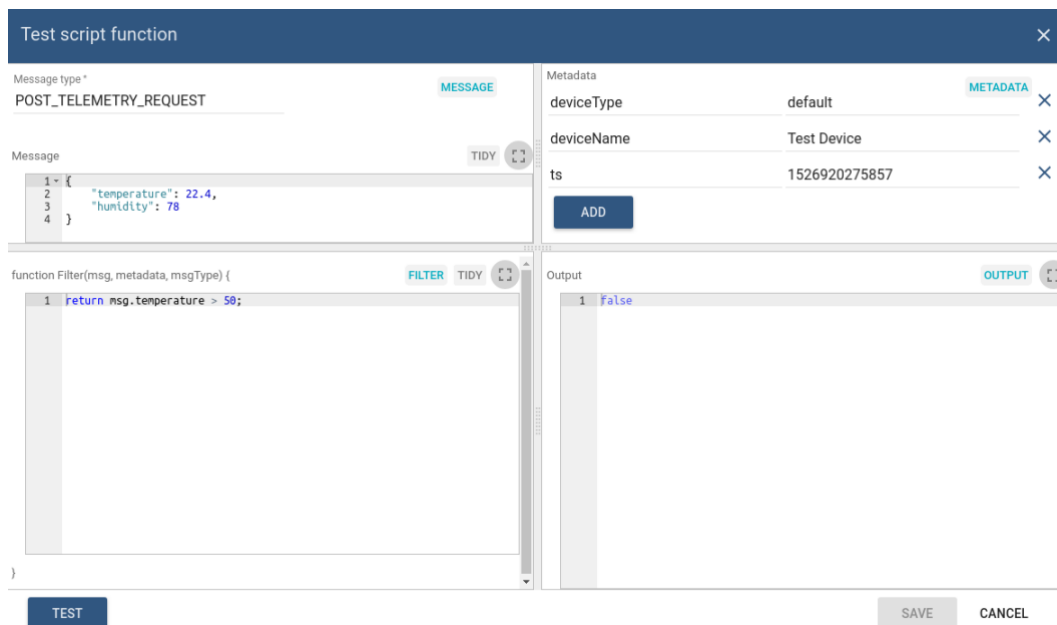


Рисунок 12 – Окно тестирования правила

Вы можете определить:

- Тип сообщения.
- Полезные данные сообщения.

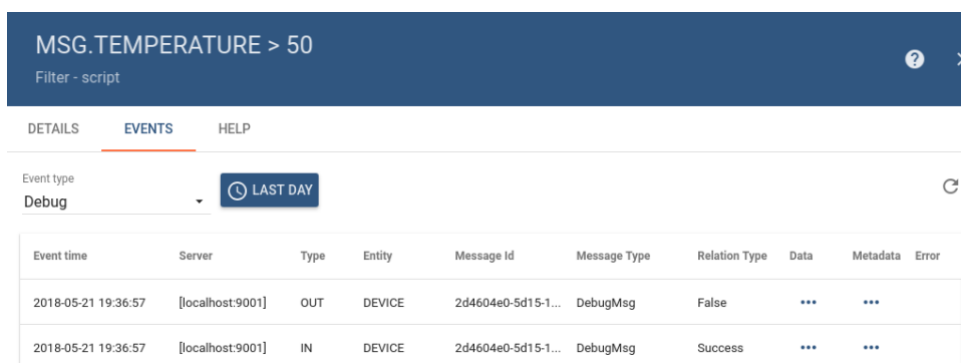
- **Метаданные.**
- Актуальный **JS-скрипт** в разделе «Фильтр».

После нажатия кнопки **TEST** вывод будет возвращен в правую секцию **вывода**.

Отладка

ThingsBoard предоставляет возможность просматривать входящие и исходящие сообщения для каждого узла правил. Чтобы включить отладку, пользователю необходимо убедиться, что в главном окне конфигурации установлен флажок «Debug mode».

После включения отладки пользователь может видеть информацию о входящих и исходящих сообщениях, если они соответствуют типам отношений. Пример приведен на рисунке 13.



Event time	Server	Type	Entity	Message Id	Message Type	Relation Type	Data	Metadata	Error
2018-05-21 19:36:57	[localhost:9001]	OUT	DEVICE	2d4604e0-5d15-1...	DebugMsg	False	***	***	
2018-05-21 19:36:57	[localhost:9001]	IN	DEVICE	2d4604e0-5d15-1...	DebugMsg	Success	***	***	

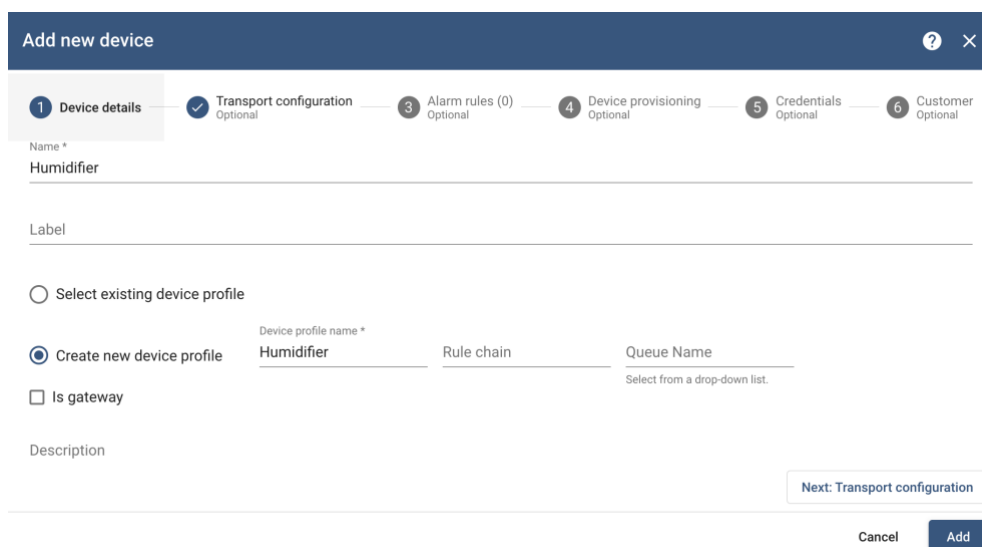
Рисунок 13 – Примеры отладочных сообщений

Пример создания цепочки правил

В качестве примера возьмем автоматический увлажнитель, который будет отправлять данные по влажности в облако, а обратно получать команды на включения и выключение.

Создание виртуального устройства

Изначально должно быть создано виртуальное устройство, через которое будет подключаться физическое устройство. Создадим виртуальный увлажнитель с наименованием Humidifier, и профилем устройства с таким же названием. Цепочку правил оставим пустой, в качестве протокола выберем MQTT.



Add new device

1 Device details | 2 Transport configuration (Optional) | 3 Alarm rules (0) (Optional) | 4 Device provisioning (Optional) | 5 Credentials (Optional) | 6 Customer (Optional)

Name *
Humidifier

Label

☐ Select existing device profile

☒ Create new device profile

Device profile name *
Humidifier

Rule chain

Queue Name
Select from a drop-down list.

☐ Is gateway

Description

Next: Transport configuration

Cancel Add

Рисунок 14 – Создание устройства

Рисунок 15 – Создание устройства

Создадим цепочку правил для контроля состояния увлажнителя. К примеру, при влажности меньше 40 он должен включаться, в противном случае, выключаться. Создать цепочку можно на вкладке Rule chains. Определим цепочку с наименованием Humidifier Control.

Рисунок 16 – Характеристики цепочки Humidifier Control

Цепочка будет содержать в себе следующий набор блоков, представленный на рисунке 17. Разберём подробнее каждый из блоков в отдельности.

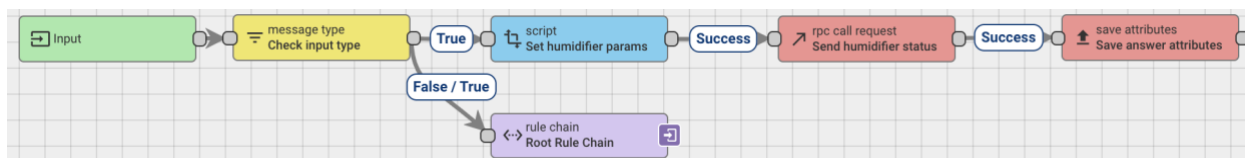


Рисунок 17 – Rule Chain Humidifier Control

Узел проверки типа приходящего сообщения

Данный узел является узлом фильтрации и позволяет определять тип приходящего в него сообщения. В данном случае при поступлении сообщения, содержащего в себе обновление телеметрии, будет происходить передача данных в узел изменения этого самого сообщения и всей сопутствующей информации. Также в любом из случаев информация будет перенаправляться в базовую цепочку правил.

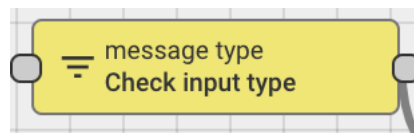


Рисунок 18 – Узел проверки типа сообщения

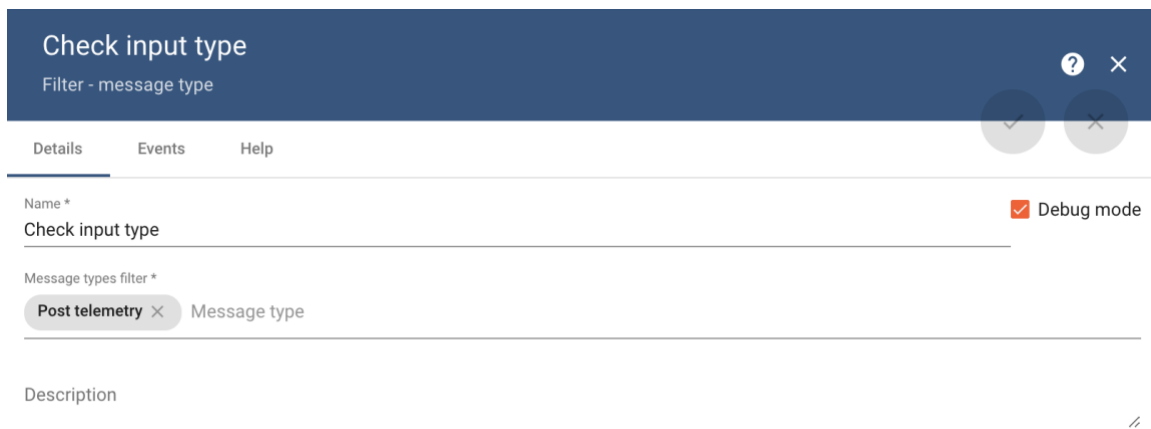


Рисунок 19 – Параметры узла проверки типа сообщения

Для фильтрации типа сообщения необходимо задать параметр Message types filter. В нашем случае необходимо в качестве типа фильтра необходимо задать Post telemetry, то есть фильтр будет пропускать только сообщения, содержащие телеметрию.

Узел перенаправления в базовую цепочку правил



Рисунок 19 – Узел перенаправления в базовую цепочку правил

Базовая цепочка правил создана для того, чтобы сохранять все приходящие данные в соответствующие разделы облака и его устройств.

Узел формирования параметров увлажнителя

Узел трансформации данных при помощи скрипта позволяет переформировать объект, содержащий в себе данные приходящего сообщения: его основную полезную нагрузку, метаданные, а также тип сообщения.

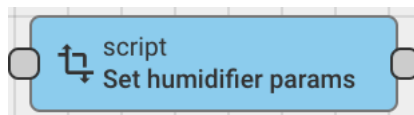


Рисунок 20 - Узел формирования параметров увлажнителя

Поведение узла описывается при помощи Java Script. Изначально в приходящей телеметрии предполагается наличие параметра уровня влажности. На основании этого параметра вычисляет новое состояние увлажнителя и формируется новый объект сообщения с этим состоянием. Данный объект содержит в себе несколько свойств: method – это наименование метода, при помощи которого можно будет идентифицировать необходимое действие на устройстве, а также свойство params, содержащее как раз состояние устройства, в которое его необходимо привести. В дальнейшем этот объект будет отправлен на конечное устройство для смены его состояния. Также изменим тип события на событие загрузки атрибутов устройства – POST_ATTRIBUTE_REQUEST. Этот параметрам пригодится нам

в будущем. На выходе узел должен возвращать объект, содержащий в себе основное сообщение, метаданные, а также тип сообщения. Полный код представлен на листинге 1.

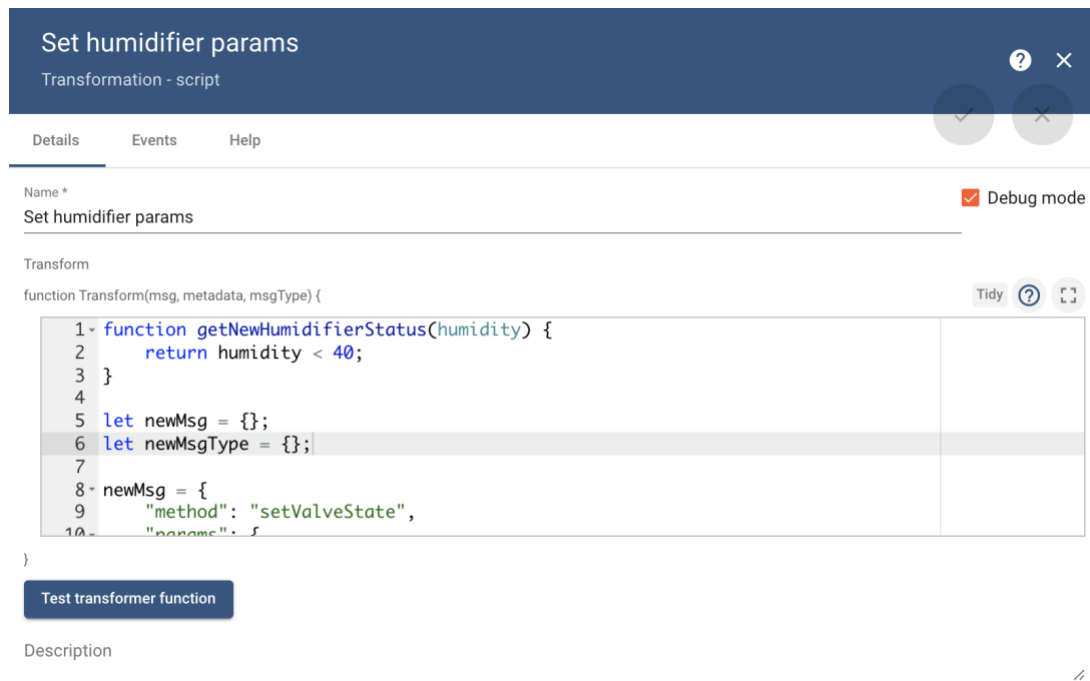


Рисунок 21 – Параметры цепочки правил

Листинг 1 – Код узла формирования параметров увлажнителя

```
function getNewHumidifierStatus(humidity) {  
  return humidity < 40;  
}  
  
let newMsg = {};  
let newMsgType = {};  
  
newMsg = {  
  "method": "setValveState",  
  "params": {  
    "state": getNewHumidifierStatus(msg.humidity)  
  }  
};  
  
newMsgType = "POST_ATTRIBUTES_REQUEST";  
  
return {msg: newMsg, metadata: metadata, msgType: newMsgType};  
}
```

Узел отправки данных на конечное устройство

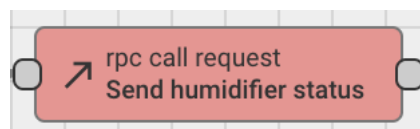


Рисунок 22 – узел отправки данных на конечное устройство

Отправляет запросы RPC на устройство и направляет ответ на следующие узлы правила. Отправитель сообщения должен быть объектом устройства, поскольку запрос RPC может быть инициирован только устройству.

В конфигурации узла есть поле Timeout, используемое для указания времени ожидания ответа от устройства. Оставим его по умолчанию равным 60 секундам.

Полезные данные сообщения должны иметь правильный формат для запроса RPC. Он должен содержать поля `method` и `params`, которые и были заданы в предыдущем узле

Если полезные данные сообщения могут содержать поле `requestId`, его значение используется для идентификации запроса RPC к устройству. Если данного свойства не будет, будет сгенерирован случайный идентификатор запроса.

Исходящее сообщение будет иметь того же отправителя и метаданные, что и входящее сообщение. Ответ от устройства будет добавлен в полезные данные сообщения.

Сообщение будет перенаправлено через цепочку отказов в следующих случаях:

- Входящее сообщение отправитель не является устройством объекта
- Во входящем сообщении отсутствуют поля метода или параметров
- Если узел не получит ответа во время настроенного тайм-аута

В противном случае сообщение будет перенаправлено через цепочку успехов.

Узел сохранения атрибутов

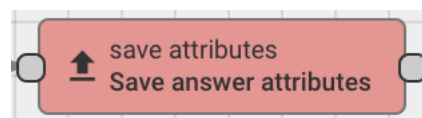


Рисунок 23 – Узел сохранения атрибутов

Узел сохраняет возвращенные в ответе на RPC запрос данные от устройства. Будем сохранять данные от устройства в виде клиентских атрибутов, для этого зададим в параметре `Entity attributes scope` значение `Client attributes`. Как раз для корректной обработки сохранения в тип сообщения ранее было записано значение `POST_ATTRIBUTE_REQUEST`.

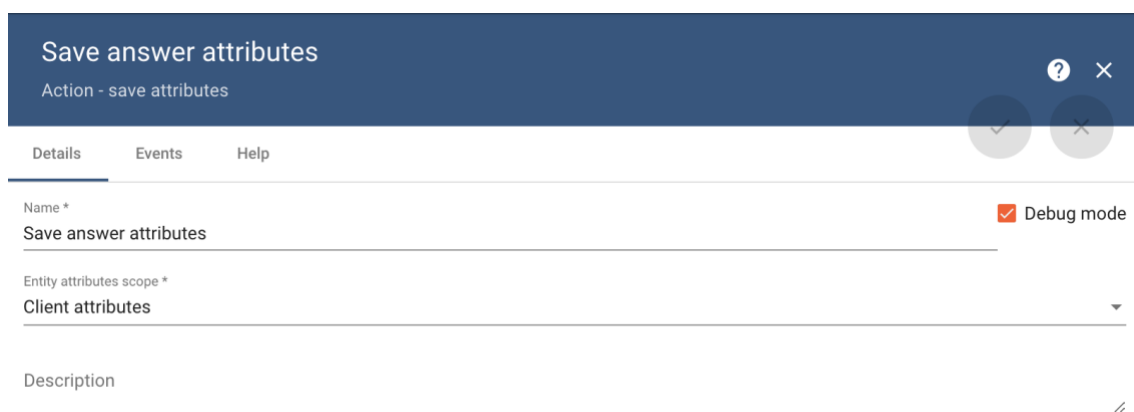


Рисунок 24 – Параметры узла сохранения атрибутов

Привязка цепочки правил к профилю

Теперь привяжем цепочку правил к соответствующему профилю, который был создан до этого – `Humidifier`. Для этого откроем на вкладке `Device Profiles` профиль `Humidifier` и в качестве параметра `Rule chain` зададим `Humidifier Control`.

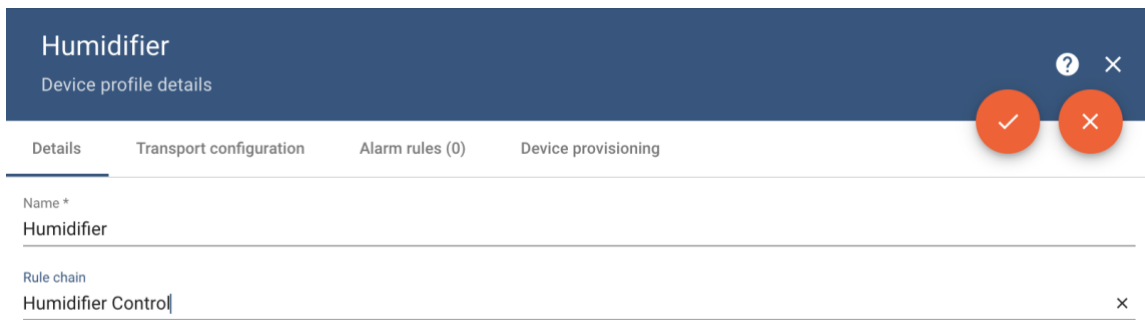


Рисунок 25 – Характеристики профиля Humidifier

Теперь проверим работоспособность правила при помощи отправки сообщений утилитами mosquitto.

Тестирование созданной цепочки

Подпишемся на топик запросов (v1/devices/me/rpc/request/+) созданного до этого виртуального увлажнителя. Воспользуемся для этого следующей командой:

```
mosquitto_sub -v -h "demo.thingsboard.io" -t  
"v1/devices/me/rpc/request/+" -u "<access_token>"
```

После чего опубликуем сообщение с телеметрией в топик устройства с параметром humidity.

```
mosquitto_pub -d -q 1 -h "demo.thingsboard.io" -t  
"v1/devices/me/telemetry" -u "<access_token>" -m '{"humidity":  
23.8}'
```

Поступление запроса к устройству можно теперь отследить в выполненной ранее команде на подписку на топики.

```
[ ~ % mosquitto_sub -v -h "demo.thingsboard.io" -t "  
v1/devices/me/rpc/request/+" -u "iGnVogfphFw8LIkJtBu"  
v1/devices/me/rpc/request/3 {"method":"setValveState","params":{"state":true}}
```

Рисунок 26 – Результаты получения сообщений в mosquitto_sub

Чтобы отправить ответ на опубликованный запрос на смену состояния, воспользуемся также утилитой mosquitto_pub. Для начала надо определить id запроса. Он идет последним топиком, в который приходит сообщение. В рассматриваемом случае это 3.

Ответы на запросы отправляются в топики v1/device/me/response. То есть для отправки ответа на созданный запрос необходимо послать сообщение в топик v1/device/me/rpc/response/3. Воспользуемся для этого следующей командой.

```
mosquitto_pub -d -q 1 -h "demo.thingsboard.io" -t  
"v1/devices/me/rpc/response/<request_id>" -u "<access_token>" -m  
{"status": 1}
```

Ответ на запрос должен содержать в себе информацию о результатах выполнения. В данном случае отправим JSON с информацией о состоянии устройства, а именно с параметром status, равным единице, для имитации подтверждения включения увлажнителя.

После этого можно проверить в облаке поступившую телеметрию и аргументы устройства, посланные в ответ на запрос. Результаты представлены на рисунках 27 и 28.

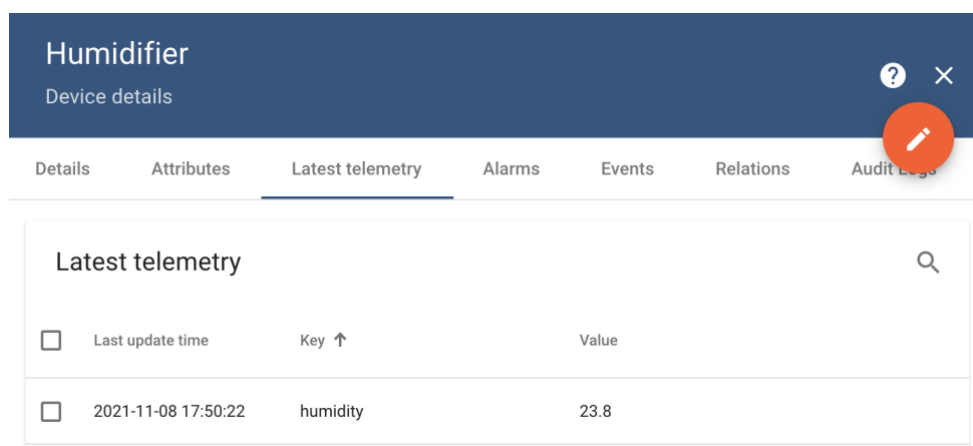


Рисунок 27 – Телеметрия облачного устройства

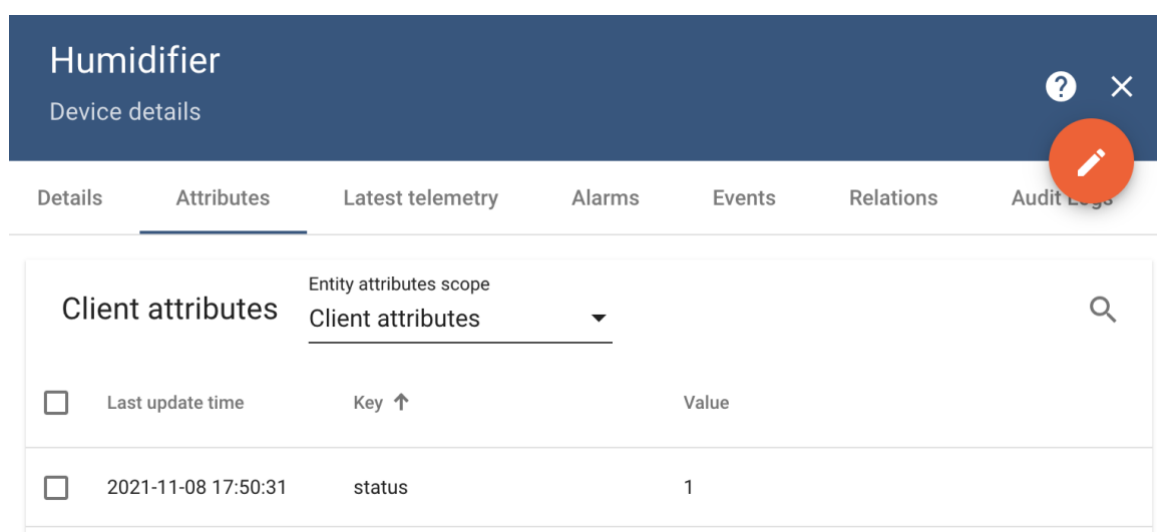


Рисунок 28 – Атрибуты облачного устройства

Как можно видеть, вся телеметрия и атрибуты зафиксировались в облачной платформе.

Задание практической работы №10

Реализуйте скрипты из практической работы №3 при помощи цепочек правил ThingsBoard, используя приведенную в методичке инструкцию.

Примечание. При ответе на RPC запрос необходимо посылать в облако состояние физического устройства, например, состояние шарового крана, в виде JSON строки: “{“valve_state”: 1}”. Или же установившийся цвет RGB ленты: “{“rgb_color”: “255;0;0”}”.

№ варианта	Сценарии
1	<ol style="list-style-type: none"> 1. Включение и выключение воды по датчику движения 2. Включение и выключение диодной ленты по кнопке
2	<ol style="list-style-type: none"> 1. Включение и выключения вентилятора по концентрации CO₂ 2. Включение, выключение и изменение звукового сигнала по кнопкам. Например, на одну кнопку значение повышается, на другую понижается, на третью происходит включение/выключение.
3	<ol style="list-style-type: none"> 1. Изменение цвета диодной ленты по концентрации CO₂ (зеленый цвет – концентрация в норме, красный – повышена)

	2. Включение и выключения световых индикаторов по кнопкам
4	1. Включение и выключение вентилятора по температуре 2. Открытие и закрытие шарового крана при одновременном нажатии двух кнопок
5	1. Включение и выключение вентилятора по датчику движения 2. Включение и выключения индикации зеленым и красным светом комбинированного датчика по кнопкам
6	1. Включение и выключение звукового сигнала по датчику силы тока 2. Включение и изменение цвета диодной ленты по кнопкам
7	1. Включение и выключение диодной ленты по датчику силы тока 2. Включение и выключение вентилятора при одновременном нажатии двух. кнопок

В отчете необходимо отразить созданные устройства, цепочки правил, а также процесс проверки правил при помощи утилит mosquitto.

Дополнительное задание практической работы №10

Часть 1. Опишите взаимодействие пользователя с интерфейсом вашего решения при помощи графической нотации, к примеру, use-case диаграмм.

Часть 2. Соберите требования к интерфейсу пользователя, включая в них:

- Какая информация и в каком виде должна предоставляться пользователю;
- Какие функции должны присутствовать в интерфейсе пользовательского приложения;
- Какого вида интерфейс необходим приложению:
 - Веб-приложение;
 - Telegram-бот;
 - и т.д.

Часть 3. На основании собранных требований разработайте макеты интерфейса вашего приложения.

Практическая работа №11 – Реакции платформ Интернета вещей на приходящие данные

Назначение реакций

Как уже было сказано ранее, платформы Интернета вещей используются для обработки данных и реализации логики работы IoT сервиса без привязки к конкретным физическим устройствам. Рассмотренный в предыдущей работе пример позволяет строить самую базовую логику по управлению устройствами, однако, никакой реакции от облачной платформы на ответы от физического устройства не предусмотрено, что не позволяет полноценно отслеживать правильность работы устройства.

IoT платформы имеют возможность отправлять оповещения (тревоги) при возникновении ошибок в ходе выполнения скриптов обработки приходящих данных. Данный механизм позволяет решить проблему отслеживания состояния физического устройства, которое передается в качестве ответа на отправляемый запрос о смене состояния. Помимо этого, можно использовать данный механизм для оповещений в случае поступления на устройство данных, выходящих за рамки допустимых значений.

Облачная платформа ThingsBoard также поддерживает подобный механизм тревог-оповещений, следовательно, рассмотрим его на примере данной платформы.

Механизм тревог платформы ThingsBoard

За работу с тревогами в ThingsBoard отвечает 2 узла правил:

- Узел создания тревоги;

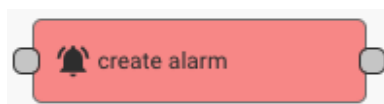


Рисунок 29 – Узел создания тревоги

- Узел отмены тревоги.

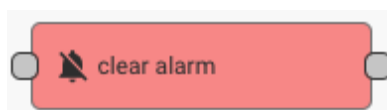


Рисунок 30 – Узел отмены тревоги

Основные концепции

Инициатор

Инициатор тревоги – это объект, который вызывает тревогу. Например, устройство А является источником сигнала тревоги, если ThingsBoard получает от него показания температуры и поднимает сигнал тревоги «Высокая температура», поскольку показание превышает пороговое значение.

Тип

Тип тревоги помогает определить основную причину тревоги. Например, «Высокая температура» и «Низкая влажность» – это два разных сигнала тревоги.

Уровень тревоги (Severity)

У каждого аварийного сигнала есть уровень: критическая, серьезная, незначительная, предупреждение или неопределенная (сортировка по приоритету в порядке убывания).

Жизненный цикл

Аварийный сигнал может быть активным или сброшенным. Когда ThingsBoard создает тревогу, он сохраняет время её начала и окончания. По умолчанию время начала и время окончания одинаковы. Если условие срабатывания сигнала тревоги повторяется, платформа обновляет время окончания. ThingsBoard может автоматически сбрасывать тревогу, когда происходит событие, соответствующее условию сброса тревоги. Условие сброса сигнала тревоги не является обязательным. Пользователь может сбросить тревогу вручную.

Помимо активного и очищенного состояния тревоги, ThingsBoard также отслеживает, подтвердил ли кто-то сигнал тревоги. Подтверждение сигнала тревоги возможно через виджет панели управления или вкладку сведений об объекте.

Подводя итог, есть 4 возможных статуса:

- Active unacknowledged (ACTIVE_UNACK) - тревога не сброшена и еще не подтверждена;
- Active acknowledged (ACTIVE_ACK) - тревога не сброшена, но уже подтверждена;
- Cleared unacknowledged (CLEARED_UNACK) - тревога уже сброшена, но еще не подтверждена;
- Cleared acknowledged (CLEARED_ACK) - тревога уже сброшена и подтверждена;

Рассмотрим пример работы с тревогами, расширив пример из прошлой практики.

Пример работы с тревогами

Добавим в цепочку правил из прошлой практической работы (рисунок 31) проверку возвращаемого ответа от устройства. Это поможет отслеживать корректность работы физического устройства при изменении его состояния.

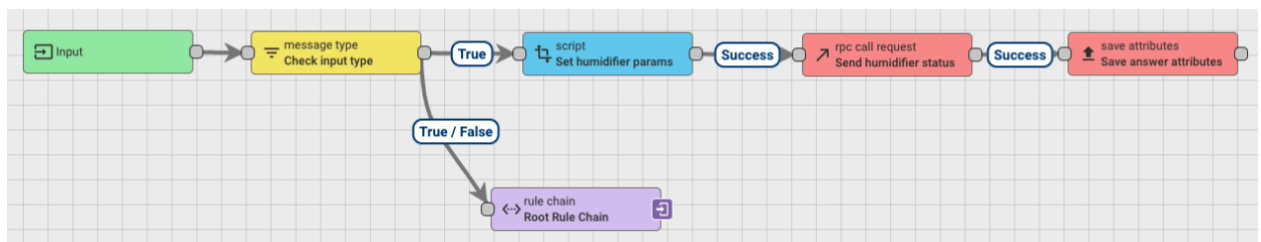


Рисунок 31 – Узел отмены тревоги

Узел сохранения атрибутов запроса

В первую очередь добавим узел сохранения атрибутов, отправляемых в запросе, чтобы впоследствии иметь возможность проверки состояния, отправляемого в запросе, с приходящем в ответе от устройства. Для этого создадим новый узел с наименованием Save request attributes и в качестве типа атрибутов выберем серверные атрибуты. Полученный узел поместим между скриптом формирования атрибутов запроса и узлом отправки запроса.

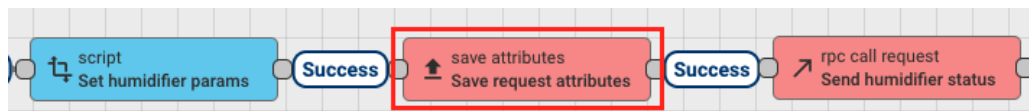


Рисунок 32 – Узел сохранения атрибутов запроса

Save request attributes

Action - save attributes

Details

Events

Help

Name *

Save request attributes

Entity attributes scope *

Server attributes

Description

Debug mode

Рисунок 33 – Параметры узла сохранения атрибутов запроса

Узел получения атрибутов запроса

Далее добавим узел получения атрибутов инициатора (originator attributes) для получения ранее сохраненных атрибутов запроса. Данному узлу можно задать выборку конкретных атрибутов из любых атрибутов устройства: клиентских, серверных или общедоступных – а также любых параметров телеметрии. Запросим из серверных атрибутов method и params, которые были сохранены до отправки запроса. Данные запишутся в метаданные цепочки правил с префиксом ss_. То есть method станет ss_method, а params станет ss_params. Итоговые параметры узла отображены на рисунке 35.

Отсоединим последний узел сохранения атрибутов ответа и вместо него присоединим только что созданный узел добавления атрибутов запроса.

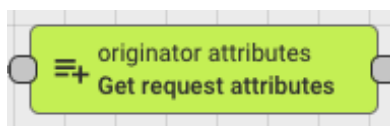


Рисунок 34 – Узел получения атрибутов запроса

Узел проверки атрибутов ответа

Теперь необходимо проверить соответствие отправляемому устройству статуса и возвращаемого от устройства статуса. Осуществим проверку при помощи скрипта из раздела фильтрующих блоков.

Сделать это можно при помощи скрипта из листинга 2.

Листинг 2. Скрипт проверки параметров

```
let request_params = JSON.parse(metadata.ss_params);

return msg.status === request_params.status;
```

Поскольку параметры для запроса были записаны в JSON формате необходимо распарсить JSON строку в объект JS для извлечения параметров при помощи функции JSON.parse.

Подключим получившийся узел к узлу получения атрибутов запроса.

Get request attributes

Enrichment - originator attributes

?

×

Details

Events

Help

Name *

Get request attributes

✓

Debug mode

✓

Tell Failure

If at least one selected key doesn't exist the outbound message will report "Failure".

Client attributes

Client attributes

Hint: use `${metadataKey}` for value from metadata, `${messageKey}` for value from message body

Shared attributes

Shared attributes

Hint: use `${metadataKey}` for value from metadata, `${messageKey}` for value from message body

Server attributes

method ×

params ×

Server attributes

Hint: use `${metadataKey}` for value from metadata, `${messageKey}` for value from message body

Latest timeseries

Latest timeseries

Hint: use `${metadataKey}` for value from metadata, `${messageKey}` for value from message body

□

Fetch Latest telemetry with Timestamp

If selected, latest telemetry values will be added to the outbound message metadata with timestamp, e.g: "temp": {"ts":1574329385897, "value":42}

Description

Рисунок 35 – Параметры узла получения атрибутов запроса

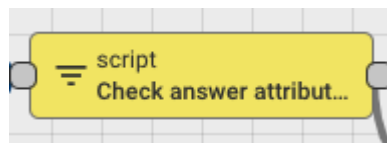


Рисунок 36 – Узел проверки атрибутов ответа

Узел формирования тревоги

Далее создадим узел активации тревоги в случае, если проверка не будет пройдена. Создадим узел create alarm и зададим ему следующий скрипт в качестве обработчика.

Листинг 3. Скрипт формирования тревоги

```

var details = {};
var request_params = JSON.parse(metadata.ss_params);

if (metadata.prevAlarmDetails) {
    details = JSON.parse(metadata.prevAlarmDetails);
    // Удаление поля prevAlarmDetails из метаданных
    delete metadata.prevAlarmDetails;
    // Теперь метаданные содержат только данные, которые были на входе
}

details.send_status = request_params.status;
details.answer_status = msg.status;

return details;

```

Скрипт записывает оба параметра – статус в запросе и статус в ответе – в детали возникающей тревоги. Помимо этого, проверяется поле prevAlarmDetails, оно содержит параметры прошлой тревоги, если она не была обработана.

Также зададим в качестве типа тревоги Wrong Answer Alarm и в качестве уровня тревоги зададим Critical. Итоговые параметры узла представлены на рисунке 38.

Присоединим полученный узел к узлу проверки с отношением False.

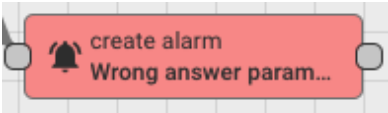


Рисунок 37 – Узел формирования тревоги

Wrong answer params alarm

Action - create alarm

Details

Events

Help

Name *

Wrong answer params alarm

Debug mode

Alarm details builder

function Details(msg, metadata, msgType) {
1 var details = {};
2 var request_params = JSON.parse(metadata.ss_params);
3
4 if (metadata.prevAlarmDetails) {
5 details = JSON.parse(metadata.prevAlarmDetails);
6 // Удаление поля prevAlarmDetails из метаданных
7 delete metadata.prevAlarmDetails;
8 // Теперь метаданные содержат только данные, которые были на входе
9 }
10 }
}

Tidy ?

Test details function

☐ Use message alarm data

☐ Use dynamically change the severity of alarm

Alarm type *

Wrong Answer Alarm

Alarm severity *

Critical

Hint: use \${metadataKey} for value from metadata, \${messageKey} for value from message body

☐ Propagate

Description

Рисунок 38 – Параметры узла формирования тревоги

Теперь присоединим узел сохранения атрибутов ответа к узлу проверки с отношением True и получим итоговый вариант цепочки правил, который представлен на рисунках 39-40.

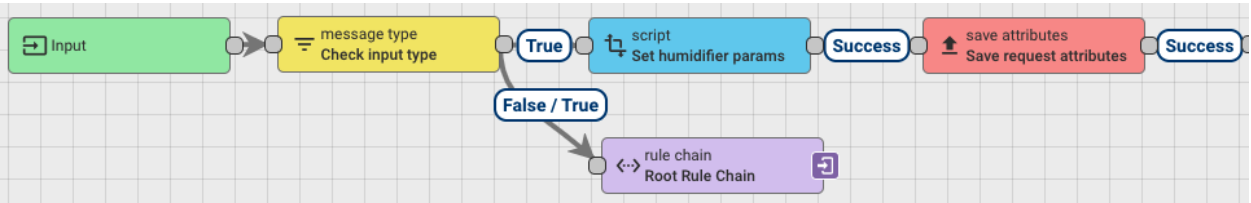


Рисунок 39 – Первая часть результирующей цепочки правил



Рисунок 40 – Вторая часть результирующей цепочки правил

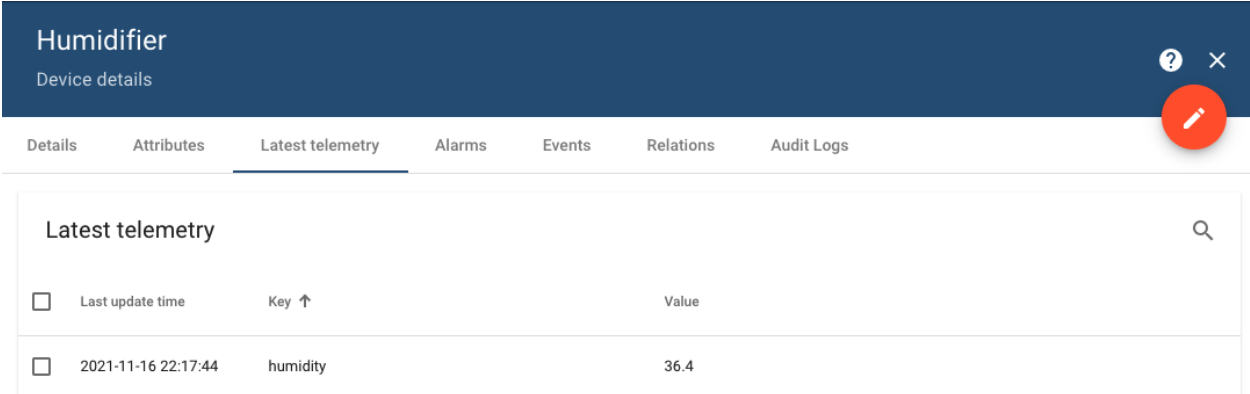
Тестирование созданной цепочки

Проведем тестирование созданной цепочки при помощи утилит mosquitto.

Для начала протестируем цепочку при получении верного ответа от устройства. Отправим данные о влажности в диапазоне, в котором должно происходить включение увлажнителя, то есть меньше 40.

```
mosquitto_pub -d -q 1 -h "demo.thingsboard.io" -t  
"v1/devices/me/telemetry" -u "<access_token>" -m '{"humidity":  
36.4}'
```

После этого в облачной платформе отобразится новая телеметрия.



Humidifier		
Device details		
Details	Attributes	Latest telemetry
Latest telemetry		
<input type="checkbox"/>	Last update time	Key
<input type="checkbox"/>	2021-11-16 22:17:44	humidity
		Value
		36.4

Рисунок 41 – Телеметрия первого теста

Также в топике запросов получим следующее сообщение:

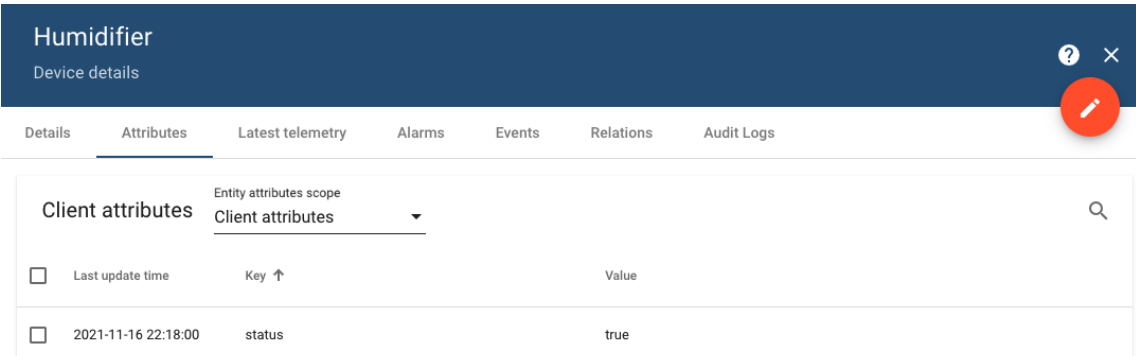
```
v1/devices/me/rpc/request/<request_id>  
{ "method": "setValveState", "params": { "status": true } }
```

Ответим на поступивший запрос также при помощи команды mosquitto_pub. Отправим верный ответ, означающий, что устройство перешло в необходимое состояние.

```
mosquitto_pub -d -q 1 -h "demo.thingsboard.io" -t  
"v1/devices/me/rpc/response/<request_id>" -u "<access_token>" -m  
'{"status": true}'
```

Примечание: обратите внимание на оформление отправляемого сообщения. Также можно экранировать символы кавычек для статуса следующим образом `"{'status': true}"`

В результате в атрибутах устройства отразится



Humidifier		
Device details		
Details	Attributes	Latest telemetry
Client attributes		
<input type="checkbox"/>	Last update time	Key
<input type="checkbox"/>	2021-11-16 22:18:00	status
		Value
		true

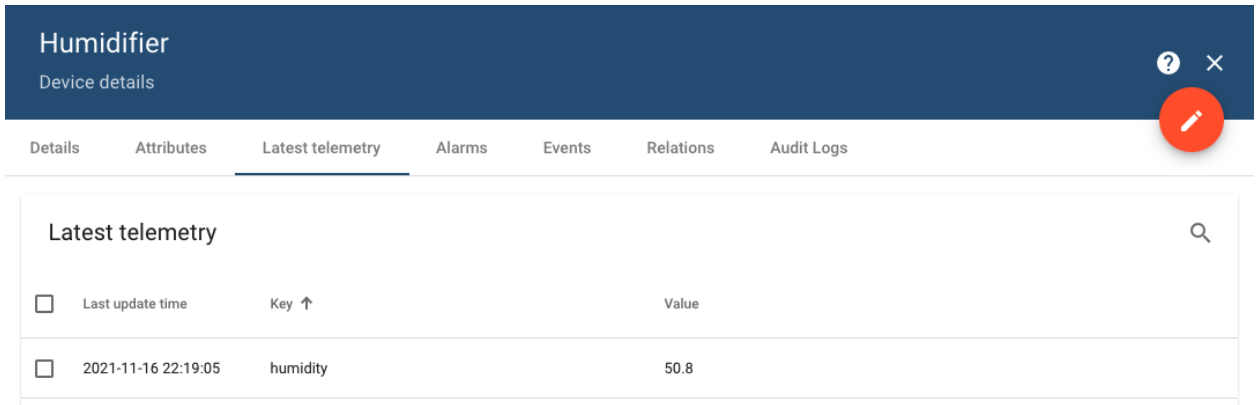
Рисунок 42 – Атрибут виртуального устройства

Проведем ещё один тест, теперь уже с отправкой неверного статуса.

Передадим данные в диапазоне, при котором увлажнитель не должен работать при помощи следующей команды.

```
mosquitto_pub -d -q 1 -h "demo.thingsboard.io" -t  
"v1/devices/me/telemetry" -u "<access_token>" -m '{"humidity":  
50.8}'
```

Проверим поступление телеметрии в облачной платформе.



Humidifier
Device details

Details Attributes Latest telemetry Alarms Events Relations Audit Logs

Latest telemetry

<input type="checkbox"/>	Last update time	Key ↑	Value
<input type="checkbox"/>	2021-11-16 22:19:05	humidity	50.8

Рисунок 43 – Телеметрия второго теста

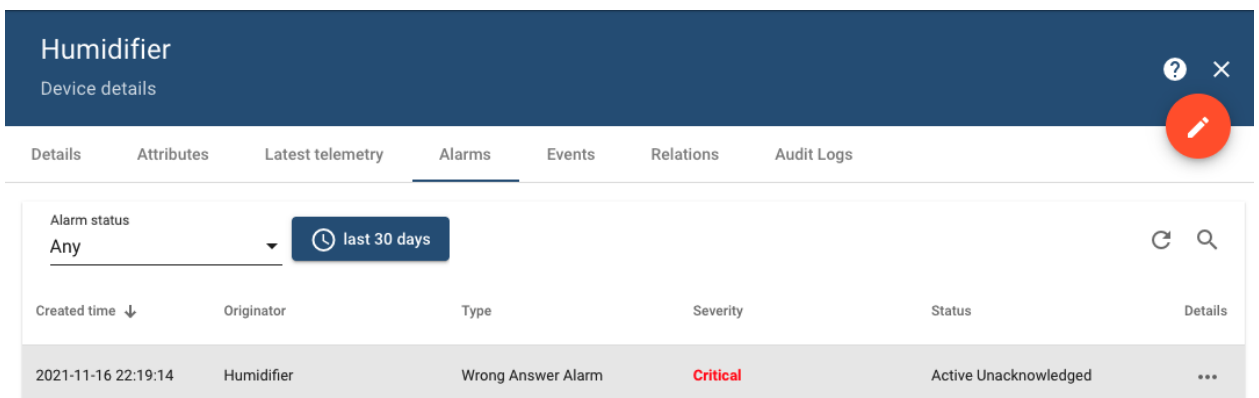
Получим следующий запрос из топика запросов устройства.

```
v1/devices/me/rpc/request/<request_id>  
{ "method": "setValveState", "params": { "status": false } }
```

В качестве ответа передадим ошибочный для данного случая статус, то есть передадим облаку статус устройства true, вместо false.

```
mosquitto_pub -d -q 1 -h "demo.thingsboard.io" -t  
"v1/devices/me/rpc/response/<request_id>" -u "<access_token>" -m  
'{"status": true}'
```

В результате получим тревогу в облаке, которая сигнализирует об ошибочном состоянии устройства. Тревоги отображаются на вкладке Alarms устройства.



Humidifier
Device details

Details Attributes Latest telemetry Alarms Events Relations Audit Logs

Alarm status
Any last 30 days

Created time ↓	Originator	Type	Severity	Status	Details
2021-11-16 22:19:14	Humidifier	Wrong Answer Alarm	Critical	Active Unacknowledged	...

Рисунок 44 – Вкладка тревог виртуального устройства

Можно посмотреть подробности о созданной тревоге кликнув по многоточию в крайнем правом столбце. Подробная информация содержит в себе в том числе детали, заданные в скрипте создания тревоги.

Alarm details

Created time

2021-11-16 22:19:14

Originator

Humidifier

Start time

2021-11-16 22:19:14

End time

2021-11-16 22:19:14

Type

Wrong Answer Alarm

Severity

Critical

Status

Active Unacknowledged

Details

1

{

2

"send_status": false,

3

"answer_status": true

4

}

Close

Acknowledge

Clear

Рисунок 45 – Окно подробностей о возникшей тревоге

Рассмотрен один вариант создания тревоги – при получении ответа от физического устройства. Однако, как уже было сказано, тревоги могут формироваться при различных обстоятельствах, к примеру при выходе параметров, собираемых устройством, за границы допустимых значений, при поступлении неизвестных параметров телеметрии и т.д.

Задание практической работы №11

Добавьте в цепочки правил из 10 практической работы формирование нескольких типов тревог:

- Для первой цепочки из варианта – тревогу при выходе приходящего параметра за допустимые границы (границы задайте самостоятельно);
- Для второй цепочки из варианта – тревогу при отсутствии ожидаемого параметра в приходящем сообщении (к примеру, в приходящем сообщении отсутствует параметр с состоянием кнопки)
- Для обеих цепочек – тревога при поступлении неверного ответа от физического устройства (это может быть неверный формат или неверное значение, отличающееся от ожидаемого).

В отчет включите обновленные цепочки правил, скрипты проверок и формирования тревог, а также результаты тестирования цепочек при помощи утилит mosquitto.

Дополнительное задание практической работы №11

Часть 1. Опишите выбранные для реализации проекта технологии с обоснованием их выбора.

Часть 2. Реализуйте пользовательский интерфейс на основании разработанных в 10 практической работе макетов интерфейса приложения. В отчет включите скриншоты итогового варианта интерфейса с кратким его описанием.

Практическая работа №12 – Отправка оповещение от облачной платформы

Использование сторонних сервисов для отправки оповещений

Механизм тревог, рассмотренный в 11 практической работе, позволяет формировать оповещения внутри облачно платформы, однако, таких оповещений может быть недостаточно. Для более быстрого реагирования на возникающие инциденты можно использовать дополнительные средства.

К примеру, одним из наиболее часто используемых вариантов является оповещение при помощи электронной почты. Сервис может использовать свой собственный почтовый сервер или же перенаправлять сообщения через сторонние сервера пересылки электронных писем, доставляя оповещения о возникающих инцидентах. Такой подход позволяет сообщать администраторам системы о возникновении сбоев даже в случае, если они в текущий момент не отслеживают состояние облачного сервиса через интерфейс облачной платформы.

Помимо электронных писем возможно применение sms сообщений для отправки оповещений. Данный подход используется для отправки наиболее срочных оповещений, поскольку позволяет быстрее реагировать на возникающие инциденты.

Помимо этого, могут применяться различные дополнительные сервисы оповещений, к примеру сервисы Amazon, такие как Amazon Simple Notification Service и т.д.

Способы оповещений на платформе ThingsBoard

Платформа ThingsBoard имеет несколько возможностей для отображения оповещений:

- Email оповещения;
- SMS оповещения;
- Amazon SNS;
- Amazon SQS;
- Google Cloud PubSub

Рассмотрим некоторые из узлов, позволяющие обеспечить передачу оповещений через сторонние сервисы.

Узел отправки Email сообщений

Данный узел предназначен для отправки электронных писем по протоколу SMTP и его защищенной версии SMTPS. Чтобы передать сообщение при помощи данного узла необходимо предварительно трансформировать сообщение цепочки правил в Email-сообщение при помощи узла To Email. Процесс формирования сообщения будет рассмотрен ниже в примере.

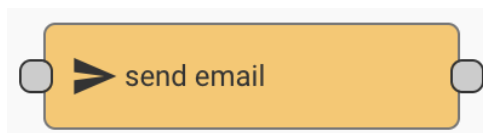


Рисунок 46 – Узел отправки Email сообщений

Для функционирования данного узла необходимо также указать используемый SMTP сервер. Если развернуть полноценный экземпляр платформы, через панель администратора можно задать системный SMTP сервер, через который будет происходить пересылка сообщений. В случае же с тестовой версией, необходимо будет вручную задавать сервер для каждого узла отправки электронных писем.

Протокол SMTP

SMTP — это простой протокол передачи почты. С английского языка переводится, как Simple Mail Transfer Protocol. SMTP сервер отвечает за отправку почтовых рассылок. Его задача, как правило, состоит из двух основных функций:

- проверка правильности настроек и выдача разрешения компьютеру, который пытается отправить электронное сообщение;
- отправка исходящего сообщения на указанный адрес и подтверждение успешной отправки сообщения. Если доставка невозможна, сервер возвращает отправителю ответ с ошибкой отправки.

Отправляя Email сообщения, SMTP-сервер отправителя устанавливает связь с тем сервером, который будет получать это сообщение. Такое "общение" происходит путем отправки и получения команд, формируя SMTP-сессию с неограниченным количеством SMTP-операций. Обязательными командами для каждой операции являются три:

- определение обратного адреса (MAILFROM)
- определение получателя Email сообщения (RCPT TO)
- отправка текста сообщения (DATA)

Определение адреса отправителя, получателя и наличие содержимого письма — это обязательные условия, без которых письмо не будет отправлено.

Узел отправки SMS

Узел формирует SMS сообщение, в которое можно записывать информацию из входящих метаданных сообщения. Узлу необходимо задать SMS-провайдера, который будет осуществлять пересылку SMS-сообщений конечным пользователям. На данный момент на платформе имеется возможность организовать пересылку при помощи двух сервисов: Amazon SNS и Twilio SMS.

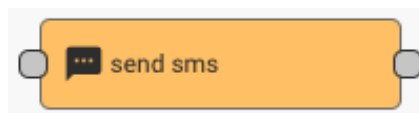


Рисунок 47 – Узел отправки sms сообщений

Как первый, так и второй сервис позволяют передавать SMS сообщения и работают по всему миру. Также Amazon SNS можно найти в качестве отдельного и связываться напрямую с данным сервисом.

Пример работы Email оповещениями

В качестве примера продолжим дополнять реализованную в прошлых практиках цепочку правил (рисунок 48-49). При возникновении тревог будем создавать Email сообщение и отправлять его на тестовый Email.

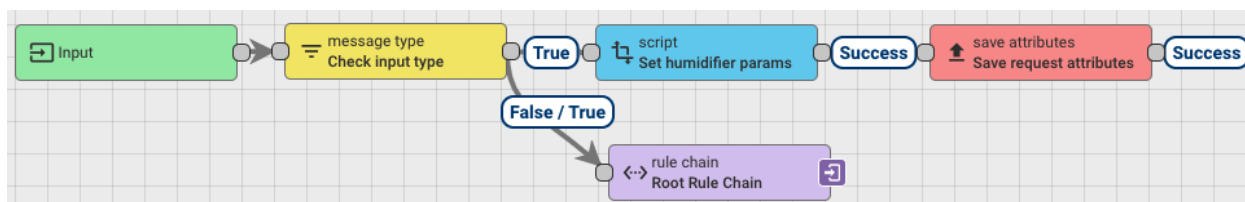


Рисунок 48 – Первая часть исходной цепочки правил

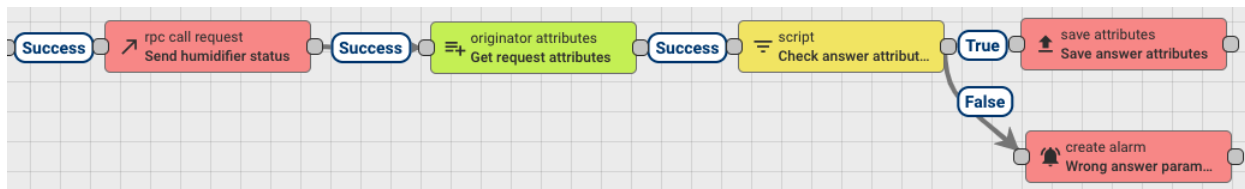


Рисунок 49 – Вторая часть исходной цепочки правил

Узел формирования Email сообщения

Во-первых, необходимо добавить узел преобразования сообщения правила в Email сообщение. Для этого используется узел To email. Данному узлу необходимо задать адрес отправителя, адрес получателя, заголовок письма, а также тела письма. Узел позволяет оперировать свойствами приходящего сообщения – можно получать свойства из сообщения при помощи конструкции `$(pror_name)` – а также приходящих метаданных – `#{pror_name}`.

Добавим данный узел в цепочку и зададим все поля. При использовании SMTP сервера Яндекс для отправки почты необходимо в качестве адреса отправителя задать тот же аккаунт, который используется для доступа к данному серверу. В качестве адреса назначения можно использовать любой адрес. Для формирования заголовка и тела письма будем использовать информацию из сообщения и метаданных, а именно: тип устройства (`deviceType`) и имя устройства (`deviceName`) из метаданных и тип тревоги (`type`) из сообщения. Итоговая конфигурация представлена на рисунках 50-51.

Рисунок 50 – Параметры узла формирования Email сообщения

Mail body type
Plain Text

Body Template *

Device \${deviceName} has alarm with type \${type}

Hint: use \${metadataKey} for value from metadata, \${messageKey} for value from message body

Description

Рисунок 51 – Параметры узла формирования Email сообщения

Будем отправлять сообщение при создании тревоги, а также при её обновлении, поэтому соединим узел создания тревоги с данным узлом отношениями Created и Updated.

Узел отправки Email сообщения

После формирования сообщения можно приступить к настройке узла отправки сообщения. В случае использования тестового варианта платформы невозможно использовать системные настройки SMTP, поэтому необходимо воспользоваться сторонним сервисом.

Предлагается использовать либо сервер Yandex, либо сервер Google. На этих двух платформах проводилось тестирование, и они точно работают.

В качестве протокола передачи данных необходимо использовать SMTPS, поскольку современные почтовые сервисы передают данные только по шифрованным протоколам. Сервер указывается в зависимости от выбранного сервиса smtp.gmail.com или же smtp.yandex.ru. Порт для обоих сервисов – 465. Таймаут можно выбрать на своё усмотрение.

После этого необходимо указать логин от аккаунта на выбранном сервисе, а в качестве пароля необходимо сгенерировать пароль для приложения. Такая функция есть у обоих сервисов:

<https://yandex.ru/support/id/authorization/app-passwords.html> и <https://support.google.com/accounts/answer/185833?hl=ru> .

Итоговую конфигурацию можно увидеть на рисунке 52.

Send email

External - send email

Details Events Help

Name *

Send email

☐ Use system SMTP settings

Protocol

SMTPS

SMTP host *

smtp.yandex.ru

SMTP port *

465

Timeout ms *

20000

☐ Enable TLS

☐ Enable proxy

Username

██████████@yandex.ru

Password

Description

☒ Debug mode

Рисунок 52 – Параметры узла отправки Email сообщений

Примечание: Обратите внимание, что при использовании SMTP сервера Yandex необходимо при формировании Email сообщения в узле To email в качестве адреса отправителя указать тот же самый логин, что и в настройках SMTP.

Результирующая цепочка правил представлена на рисунках 53-54.

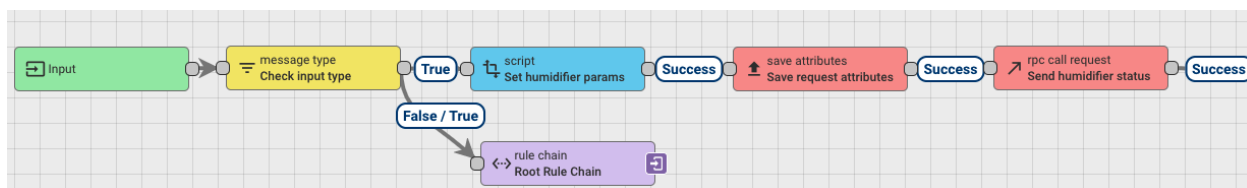


Рисунок 53 – Первая часть результирующей цепочки правил

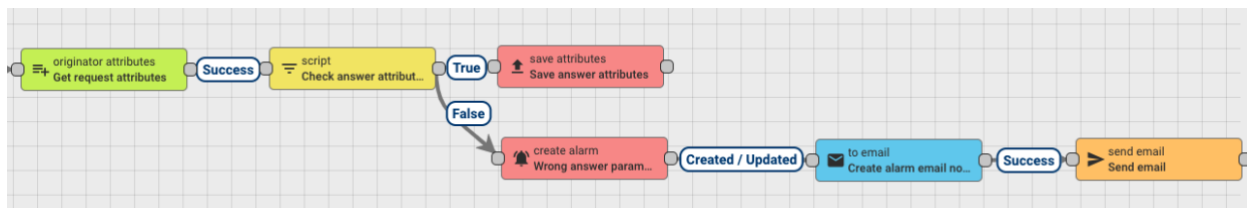


Рисунок 54 – Вторая часть результирующей цепочки правил

Тестирование созданной цепочки

Проведем тестирование созданной цепочки при помощи утилит mosquitto.

Начнем опять с отправки данных об уровне влажности. Передадим значение, при котором увлажнитель должен быть активирован.

```
mosquitto_pub -d -q 1 -h "demo.thingsboard.io" -t
"v1/devices/me/telemetry" -u "<access_token>" -m '{"humidity":
28.3}'
```

Получим запрос следующего вида из топика запросов устройства.

```
v1/devices/me/rpc/request/<request_id>
{"method":"setValveState","params":{"status": true}}
```

В качестве ответа передадим ошибочный для данного случая статус, то есть передадим облаку статус устройства false, вместо true.

```
mosquitto_pub -d -q 1 -h "demo.thingsboard.io" -t
"v1/devices/me/rpc/response/<request_id>" -u "<access_token>" -m
'{"status": false}'
```

Как ожидалось, в результате в облаке сформировалась тревога о несоответствии статуса ответа статусу запроса.

Created time	Originator	Type	Severity ↑	Status	Details
2021-11-21 17:43:47	Humidifier	Wrong Answer Alarm	Critical	Active Unacknowledged	...

Рисунок 55 – Созданная в облаке тревога

Также, помимо возникшей в облаке тревоги можно обнаружить на настроенной почте письмо с сообщением о возникновении тревоги.



Рисунок 56 – Окно подробностей о возникшей тревоге

Email сообщения могут отправляться не только в результате возникновения тревог, но и при любых других событиях на платформе. Вне зависимости от сценария использования Email сообщения позволяют повысить скорость реагирования на любые происходящие в облачной платформе события.

Задание практической работы №12

Реализуйте отправку Email сообщений из облачной платформы при возникновении тревог на узлах, созданных в практической работе №11. В качестве SMTP сервера для пересылки сообщений предлагается использовать Yandex и Google, поскольку они были протестированы. Однако, можно использовать любой другой понравившийся сервис.

В отчет включите обновленные цепочки правил, параметры узлов пересылки сообщений, а также результаты тестирования цепочек при помощи утилит mosquito и скриншоты приходящих электронных писем.

Дополнительное задание практической работы №12

Часть 1. Проведите тестирование разрабатываемого приложения. Опишите процесс тестирования, какие библиотеки, сервисы и инструменты использовали. Отдельно опишите, каким образом проводили тестирование пользовательского интерфейса.

Часть 2. Опишите процесс развертывания реализуемого приложения. Какими сервисами и инструментами пользовались при развертывании.

Требования к отчету по ПР №9-12:

По итогу выполнения практических работ №9-12 необходимо оформить единый отчёт, включающий:

1. Титульный лист;
2. Оглавление;
3. Созданные виртуальные устройства, скриншоты процесса отправки данных в облачную платформу, скриншоты полученных данных в облачной платформе для ПР №9;
4. Созданные устройства, а также цепочки правил с результатами их тестирования для ПР № 10;
5. Цепочки правил с добавленными узлами тревог, скриншоты параметров добавленных узлов со скриптами проверок и тревог, результаты тестирования при помощи утилит mosquito для ПР №11;
6. Цепочки правил с узлами отправки электронных писем, скриншоты параметров добавленных узлов, процесс тестирования отправки сообщений, получаемые письма на электронной почте для ПР №12;
7. Отчет о проекте – результаты выполнения всех дополнительных заданий ПР №9-12;
8. Выводы о проделанной работе.

Отчет по практическим работам необходимо загрузить в СДО (в случае каких-либо технических проблем отчет необходимо выслать на почту преподавателя в домене *mirea.ru*)

Литература для изучения:

1. Документация на облачную платформу ThingsBoard: <https://thingsboard.io/docs/>
2. Подключение по протоколу MQTT к платформе ThingsBoard: <https://thingsboard.io/docs/reference/mqtt-api/>
3. Гайд по работе с платформой ThingsBoard: <https://thingsboard.io/docs/getting-started-guides/helloworld/?connectdevice=mqtt-linux>
4. Документация о цепочках правил на платформе ThingsBoard: <https://thingsboard.io/docs/user-guide/rule-engine-2-0/overview/>
5. Базовый пример формирования цепочек правил: <https://thingsboard.io/docs/user-guide/rule-engine-2-0/re-getting-started/>
6. Архитектура движка правил: <https://thingsboard.io/docs/user-guide/rule-engine-2-0/architecture/>
7. Концепция тревог платформы ThingsBoard: <https://thingsboard.io/docs/user-guide/alarms/>
8. Описание узла создания тревог: <https://thingsboard.io/docs/user-guide/rule-engine-2-0/action-nodes/#create-alarm-node>
9. Гайд по созданию тревог на платформе ThingBoard: <https://thingsboard.io/docs/user-guide/rule-engine-2-0/tutorials/create-clear-alarms/>
10. Описание узла отправки Email сообщений: <https://thingsboard.io/docs/user-guide/rule-engine-2-0/external-nodes/#send-email-node>
11. Статья о протоколе SMTP: <https://selectel.ru/blog/smtp-protocol/>
12. Гайд по отправке электронных писем на платформе ThingsBoard: <https://thingsboard.io/docs/user-guide/rule-engine-2-0/tutorials/send-email/>