Elixir для интернета вещей

BACKEND

Вадим Саттаров
Elixir-разработчик

План выступления

1. Почему выбран Elixir, возможные сценарии применения.
2. Выбор железа, софта и протокола сообщения между ними.
3. Реализация протокола со стороны железа и сервера.
4. Реализация веб-интерфейса.
5. Ещё несколько слов об Elixir для IoT.
6. Заключение.

ВЕЩАЕТ

Вадим Саттаров

Elixir/Erlang разработчик в FunBox

СВЯЗАТЬСЯ СО МНОЙ

v.sattarov

mayosx

КОД ЭТОГО ДОКЛАДА

https://github.com/sattarovvadim/iotmeetup

Почему выбран Elixir, Возможные сценарии применения

Функциональная семантика

1

Тонкие процессы

2

Высокая надёжность приложения в целом, OTP

3

Сценарии применения «Интернета вещей»

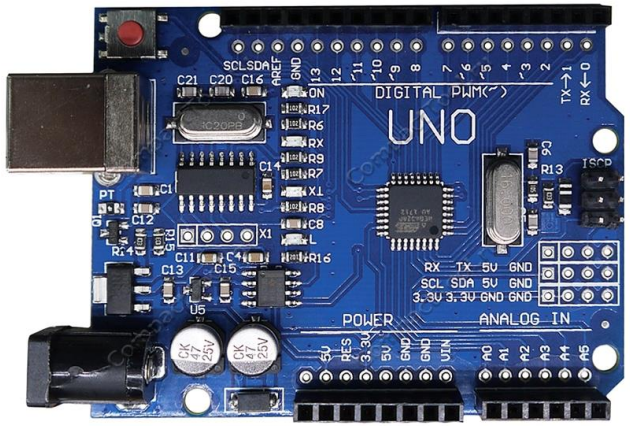
- Автоматический сбор и отправка показаний счётчиков.
- Защита от протечек воды и газа, пожарная сигнализация.
- Защитная сигнализация.
- Управление климатом.
- Управление освещением, электроворотами, поливом и другими устройствами на участке.
- Удалённый контроль за состоянием электрооборудования в доме.
- Автополив цветов, кормление домашних животных.
- Управляемые шторы.

Чем можно управлять:

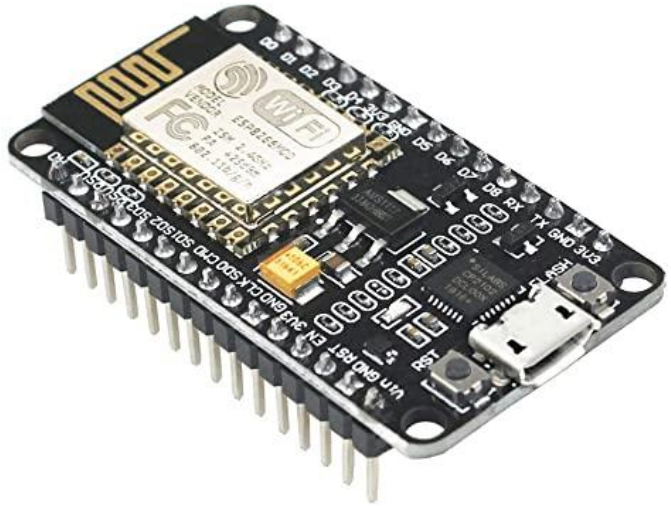
- Нагрузка любого масштаба, в том числе на 220В, 380В.
- Сервоприводы, электромагнитные устройства, двигатели.
- ИК-светодиоды.
- Индикация на ЖКИ, газоразрядных и прочих индикаторах.
- Цифровые устройства, передача файлов, видео- и аудиосигнала.

Выбор железа, софта и протокола сообщения между ними

Микроконтроллеры:

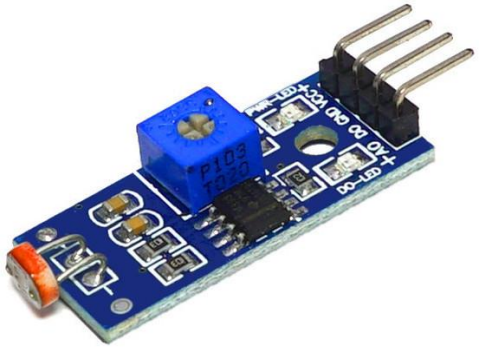


Arduino UNO

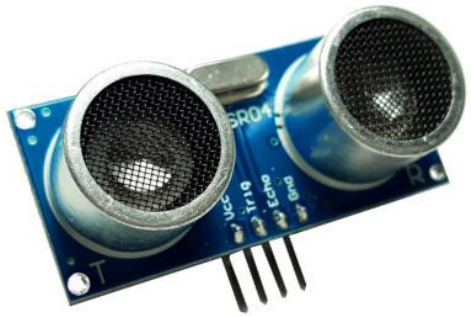


ESP8266 в виде
NodeMCU v2 Amica

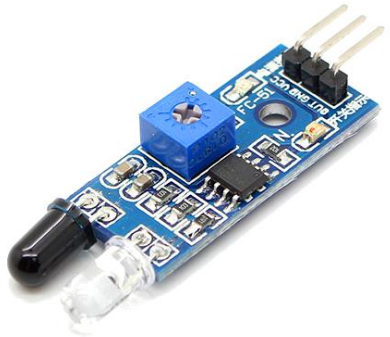
Сенсоры:



Датчик освещённости
Arduino Shield



Акустический дальномер
Arduino Shield



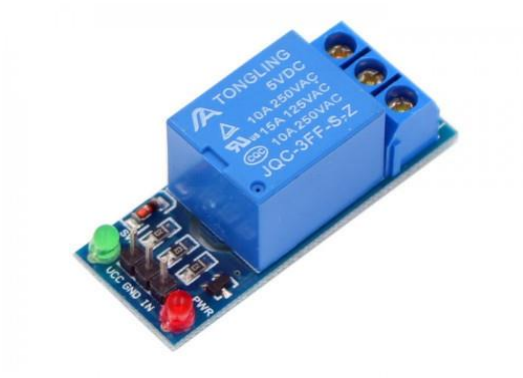
Оптический датчик препятствия
Arduino Shield

Датчик движения
Arduino Shield

Датчик температуры
Термистор

Потенциометр —
переменный резист

Исполнительные механизмы:



Реле нагрузки
Arduino Shield

Сервопривод SG90

Светодиод, резистор

ЖК индикатор 1602A

Возможные варианты архитектуры

1. Вся логика зашита в микроконтроллер.

- Для монолитных устройств, где важна скорость обработки данных.
- Относительно простая логика работы.
- МК слабы для сложной логики.
- Сложности синхронизации данных при потере питания.

2. Локальный HTTP-сервер.

- Все данные остаются в локальном хранилище.
- Необходимо постоянно включённое устройство или компьютер.
- Сложности с синхронизацией данных при отключении питания.
- HTTP прост, но изначально создан не для IoT, работает в одну сторону.

3. **Локальный MQTT-брокер.**

4. **Свой MQTT-брокер, поднятый в облаке.**

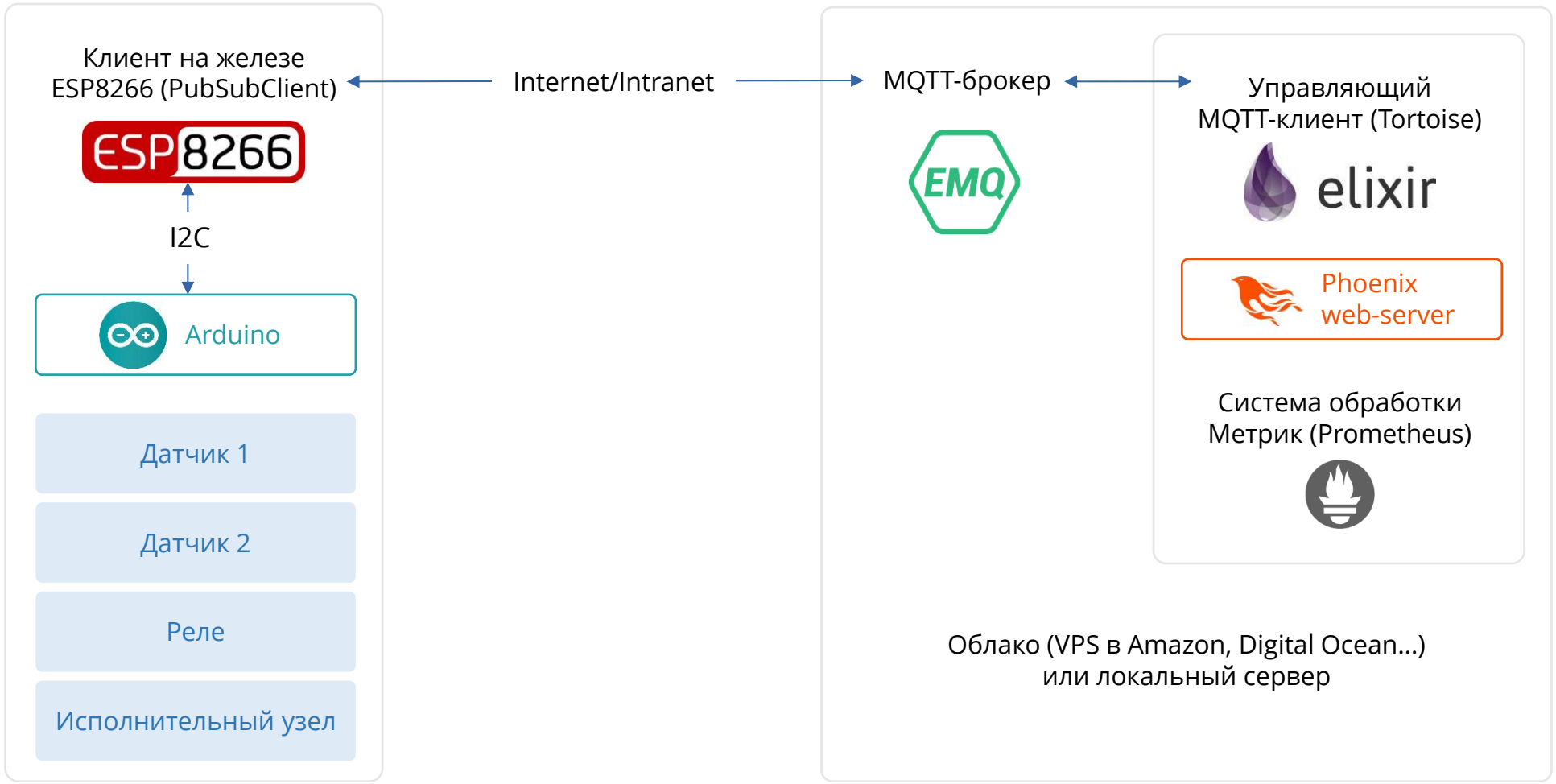
   - Поднят 99,9% времени, всегда и везде доступен, данные хорошо сохранены.
   - Необходимо обеспечить авторизацию и шифрование каналов.
   - Ежемесячные платежи за сервер.

5. Сторонний MQTT-брокер по модели SaaS.

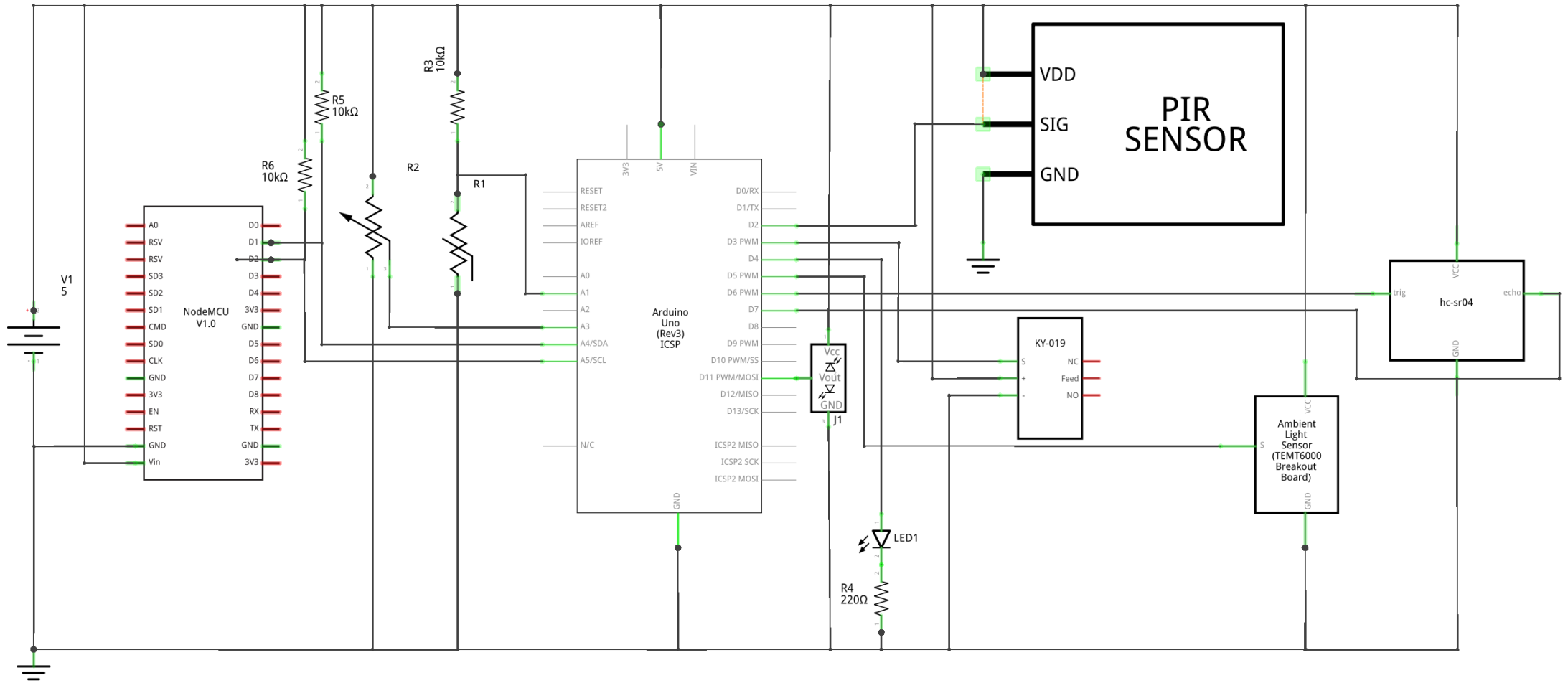
- Намного меньше возможностей для организации необходимой структуры.
- Ежемесячные платежи.
- Добавление устройств и организация логики через браузер или приложение.
- Для более сложных систем необходимо программировать МК.

Выбранная архитектура

Вариант 3 — локальный MQTT-брокер с управляющим клиентом на Elixir.



Принципиальная схема



fritzing

Реализация протокола со стороны железа и сервера

MQTT — это протокол обмена сообщениями поверх TCP по шаблону издатель-подписчик (pub/sub)

Особенности MQTT:

- Асинхронный протокол.
- Компактные сообщения (оверхед от двух байт на один пакет).
- Работа в условиях нестабильной связи на линии передачи данных.
- Поддержка нескольких уровней качества обслуживания (QoS).
- Легкая интеграция новых устройств.
- Продается уже очень много устройств с поддержкой MQTT.

Отличное описание протокола на русском языке:

https://ipc2u.ru/articles/prostye-resheniya/chto-takoe-mqtt

Топики MQTT

Все сообщения клиенты отправляют и принимают из «топиков».
Примеры топиков MQTT:

```
# Датчик температуры на кухне
home/kitchen/temperature
```

```
# Датчик температуры в спальне
home/sleeping-room/temperature
```

```
# Датчик освещенности на улице
home/outdoor/light
```

```
# Одноуровневый wildcard
home/+/temperature
```

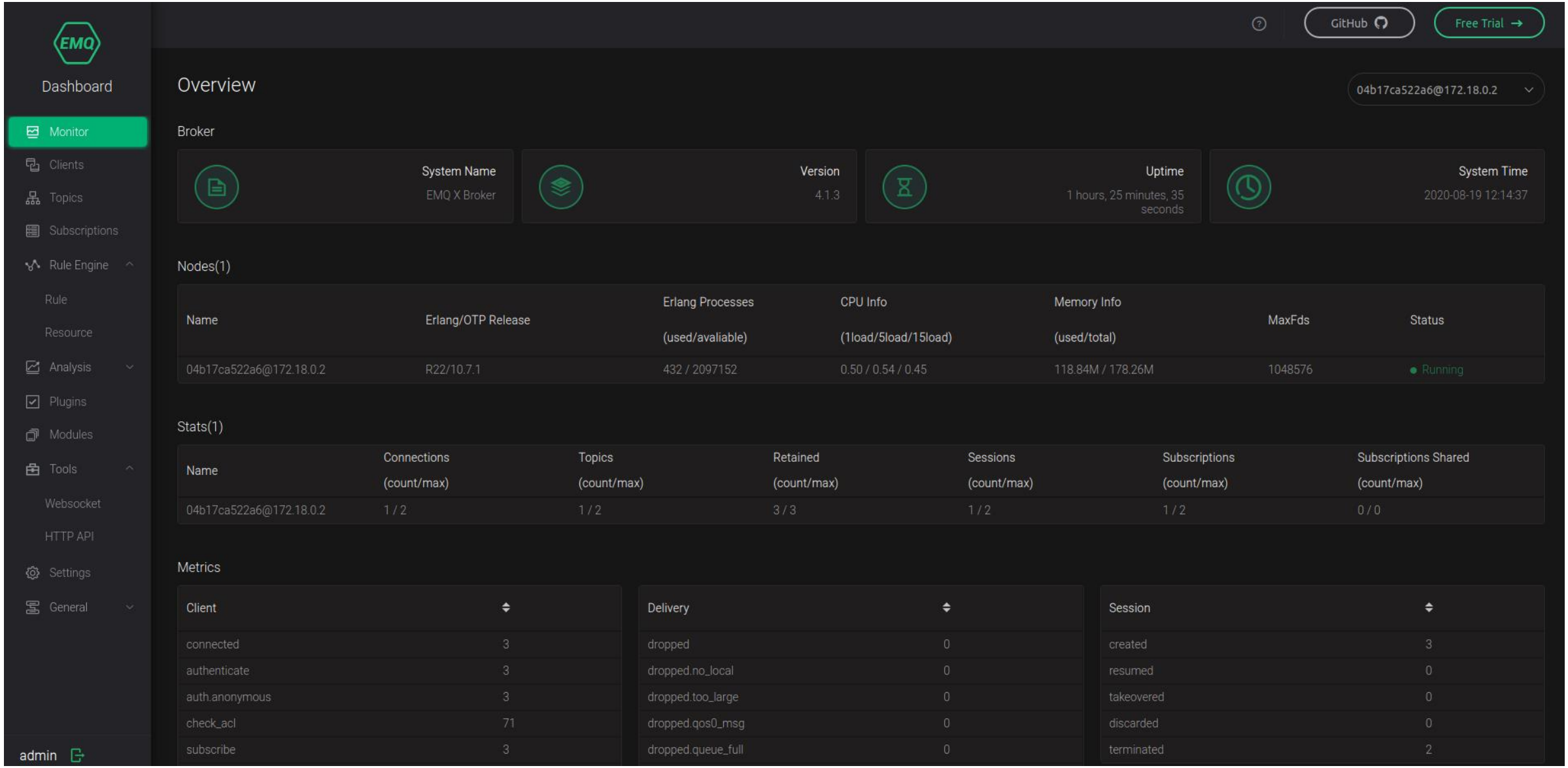
```
#многоуровневый wildcard
home/#
```

Брокер в нашей системе

EMQX — брокер, написанный на Erlang.

```
docker run -d --name emqx -p 1883:1883 -p 8083:8083 -p 8883:8883 -p 8084:8084 -p 18083:18083 emqx/emqx
```

http://127.0.0.1:18083 - web-панель брокера


EMQ
Dashboard
Monitor
Clients
Topics
Subscriptions
Rule Engine
Rule
Resource
Analysis
Plugins
Modules
Tools
Websocket
HTTP API
Settings
General
admin
GitHub
Free Trial
Overview
04b17ca522a6@172.18.0.2
Broker
System Name
EMQ X Broker
Version
4.1.3
Uptime
1 hours, 25 minutes, 35 seconds
System Time
2020-08-19 12:14:37
Nodes(1)
Name
Erlang/OTP Release
Erlang Processes (used/avaliable)
CPU Info (1load/5load/15load)
Memory Info (used/total)
MaxFds
Status
04b17ca522a6@172.18.0.2
R22/10.7.1
432 / 2097152
0.50 / 0.54 / 0.45
118.84M / 178.26M
1048576
Running
Stats(1)
Name
Connections (count/max)
Topics (count/max)
Retained (count/max)
Sessions (count/max)
Subscriptions (count/max)
Subscriptions Shared (count/max)
04b17ca522a6@172.18.0.2
1 / 2
1 / 2
3 / 3
1 / 2
1 / 2
0 / 0
Metrics
Client
connected 3
authenticate 3
auth.anonymous 3
check_acl 71
subscribe 3
Delivery
dropped 0
dropped.no_local 0
dropped.too_large 0
dropped.qos0_msg 0
dropped.queue_full 0
Session
created 3
resumed 0
takeovered 0
discarded 0
terminated 2

Реализация «управляющего клиента» на Elixir

Клиент — обычное Phoenix-приложение.

MQTT-клиент для Elixir — Tortoise — https://hexdocs.pm/tortoise/Tortoise.html

```
# Запустить воркер для wildcard-подписки на все сенсоры
 Tortoise.Supervisor.start_child(
     client_id: "elixir-main-client",
     handler: {Tortoise.Handler.Logger, []},
     server: {Tortoise.Transport.Tcp, host: 'localhost', port: 1883},
     subscriptions: [{"/iotmeetup/espdevice/+", 0}]
 )

 # Запустить воркер для подписки на сенсор освещенности
 Tortoise.Supervisor.start_child(
     client_id: "elixir-main-client",
     handler: {Server.IsLightHandler, []},
     server: {Tortoise.Transport.Tcp, host: 'localhost', port: 1883},
     subscriptions: [{"/iotmeetup/espdevice/is_light", 0}]
 )
```

Вывод консоли при приходе сообщения из платы

Стандартный логер Tortoise.Handler.Logger просто логирует приходящие уже разобранные MQTT-сообщения.

```
vadim@vadim-pc:~/projects/iotmeetup/server$ mix phx.server
[info] Running ServerWeb.Endpoint with cowboy 2.8.0 at 0.0.0.0:4000 (http)
[info] Access ServerWeb.Endpoint at http://localhost:4000
[info] Initializing handler
[info] Connection has been established
[info] Subscribed to /iotmeetup/espdevice/#
[info] /iotmeetup/espdevice/handshake "Hello from ESP, here is some handshake info"
[info] /iotmeetup/espdevice/is_motion <<1>>
[info] /iotmeetup/espdevice/is_barrier <<1>>
[info] /iotmeetup/espdevice/is_barrier <<0>>
[info] /iotmeetup/espdevice/is_motion <<0>>
[info] /iotmeetup/espdevice/is_motion <<1>>
[info] /iotmeetup/espdevice/is_light <<1>>
[info] /iotmeetup/espdevice/is_light <<0>>
[info] /iotmeetup/espdevice/is_motion <<0>>
[info] /iotmeetup/espdevice/is_barrier <<1>>
```

Отправка MQTT-сообщения из микроконтроллера

Библиотека для Arduino — PubSubClient — https://github.com/knolleary/pubsubclient

```
// теперь проверим, что изменилось, и отправим для изменившихся значений
// пуш в mqtt-топик
for (char i = 0; i < numNodes; i++)
{
    if (old_data[i] == data[i])
    {
        continue;
    }
    Node node = nodes_list.get_node_by_ind(i);
    if (!node.fire_publish)
    {
        continue;
    }
    char topic[32];
    snprintf(topic, 32, "iotmeetup/espdevice/%s", node.key);
    const uint8_t payload[1] = { data[i] };
    client.publish(topic, payload, 1);
}
```

Обработка сообщений из топика в микроконтроллере

```
void on_mqtt_receive(char *topic, byte *payload, unsigned int length)
{
    char value = payload[0];

    // арифметика указателей, чтобы получить кусок строки, содержащий имя узла/команды
    char *cmd = topic + strlen("iotmeetup/espdevice/command/");

    char command = nodes_list.set_value(cmd, value);

    handle_command(command, value);
}
```

Обработка сообщений из MQTT-топика в управляющем клиенте

```
def handle_message(["iotmeetup", "espdevice", node], <<data>>, state) do
  key = String.to_existing_atom(node)

  new_state = Map.put(state, key, data)
  BoardData.store(new_state)
  IotMeetupChannel.broadcast_data(node, Map.put(%{}, key, data))

  {:ok, new_state}
end

def handle_message(topic, publish, state) do
  Logger.info("Unknown topic #{Enum.join(topic, "/")} #{inspect(publish)}")
  {:ok, state}
end
```

Отправка команд в MQTT из управляющего клиента

```
def command_to_board(node, value) do
  Tortoise.publish(
    Application.fetch_env!(:server, :client_id),
    "iotmeetup/espdevice/command/#{Atom.to_string(node)}",
    prepare_payload(value)
  )
end
```

Реализация веб-интерфейса

Реализован
на ReactJS

1

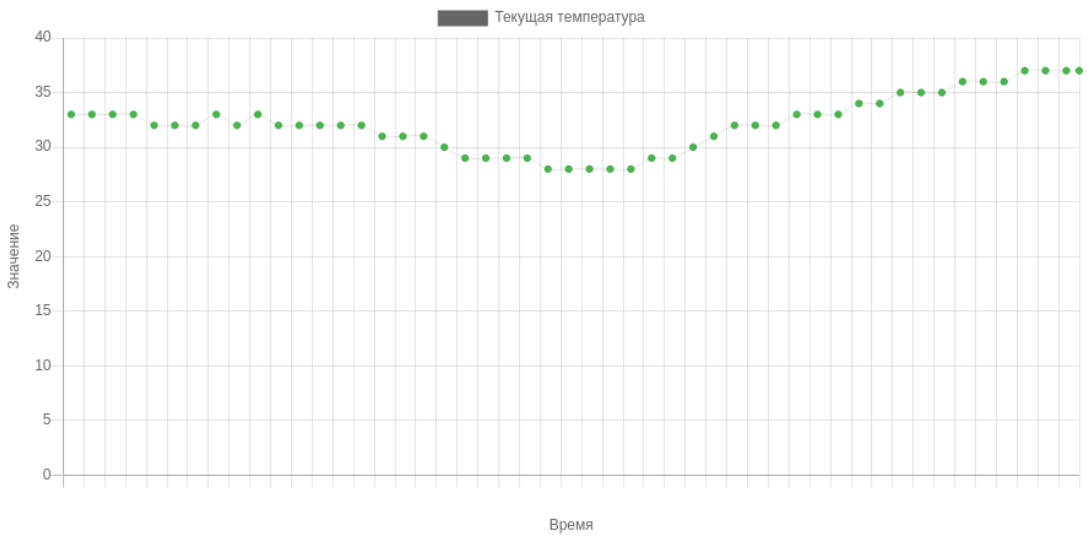
Обменивается
данными с бекендом
с помощью Phoenix
Channel посредством
веб-сокетов

2

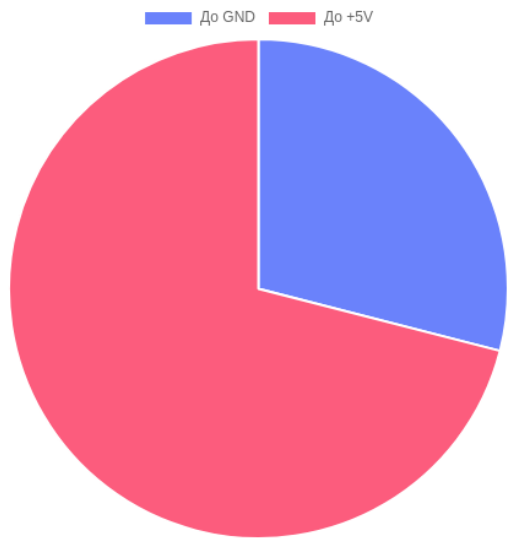
Страница обновляется
в реальном времени
по приходе данных
с платы

3

ТЕМПЕРАТУРА



ПОТЕНЦИОМЕТР



Свет включен
Датчик освещенности

Движения нет
Датчик движения

Препятствие есть
Датчик препятствия

Датчик расстояния: 52см

Светодиод:

220 Вольт:

Сервопривод

Ещё несколько слов об Elixir для IoT

Несколько self-hosted MQTT-сервисов:

https://mosquitto.org — самый популярный self-hosted брокер, написан на C.

Написанные на Erlang брокеры:

https://vernemq.com — на гитхабе — https://github.com/vernemq/vernemq

https://www.emqx.io — на гитхабе — https://github.com/emqx/emqx

https://github.com/alekras/erl.mqtt.server

https://www.rabbitmq.com — поддерживает MQTT
с плагином https://www.rabbitmq.com/mqtt.html

Облачные MQTT-брокеры:

https://cloudmqtt.com — раньше был бесплатный пробный тариф с 5 активными подключениями, но недавно его убрали.

https://cloud.yandex.ru/docs/iot-core/concepts/mqtt-properties — IoT Яндекса.

https://en.wikipedia.org/wiki/Comparison_of_MQTT_implementations — сравнение брокеров.

MQTT-клиенты для разных платформ:

https://play.google.com/store/apps/details?id=ru.esp8266.iotmanager&hl=ru — Android - приложение с приятным интерфейсом

https://www.hivemq.com/blog/seven-best-mqtt-client-tools/ — несколько инструментов для работы с MQTT для компьютера

https://github.com/Johann-Angeli/wireshark-plugin-mqtt - плагин для WireShark

Проекты реализации встраиваемых систем на базе Erlang/Elixir

https://nerves-project.org — нацелен на создание встраиваемой экосистемы на базе Elixir; https://hexdocs.pm/nerves/targets.html — тут указаны целевые платформы для Nerves.

https://grisp.org — ещё более радикальный подход, предполагающий запуск ERTS напрямую на железе (bare metal Erlang system).

https://github.com/bettio/AtomVM и https://github.com/cloudozer/ling — попытки создать компактную версию ERTS, пригодную для применения в микроконтроллерах и слабых SoC/SoM.

Возможно, в следующем докладе мы попробуем запустить Erlang на RaspberriPi.

Экономика процесса

ESP8266 — голый 110 рублей, NodeMCU — от 300 до 400 рублей.

Arduino UNO — 500 рублей.

Arduino NANO — 200 рублей.

Сенсоры-шилды, сервопривод — примерно по 250 рублей.

Термистор, диоды, резисторы — от 20 до 50 рублей за штуку.

Электровентили для воды на 220В — от 1 000 до 3 000 рублей за штуку.

Полезные ссылки и использованные материалы

Этот митап:
https://github.com/sattarovvadim/iotmeetup

Статьи об MQTT:
https://habr.com/ru/post/463669
https://ipc2u.ru/articles/prostye-resheniya/chto-takoe-mqtt

Список библиотек MQTT для разных технологий:
https://github.com/mqtt/mqtt.github.io/wiki/libraries

Собирательная таблица MQTT-брокеров:
https://github.com/mqtt/mqtt.github.io/wiki/server-support

Практически любой материал:
https://www.google.ru/search?q=elixir+iot