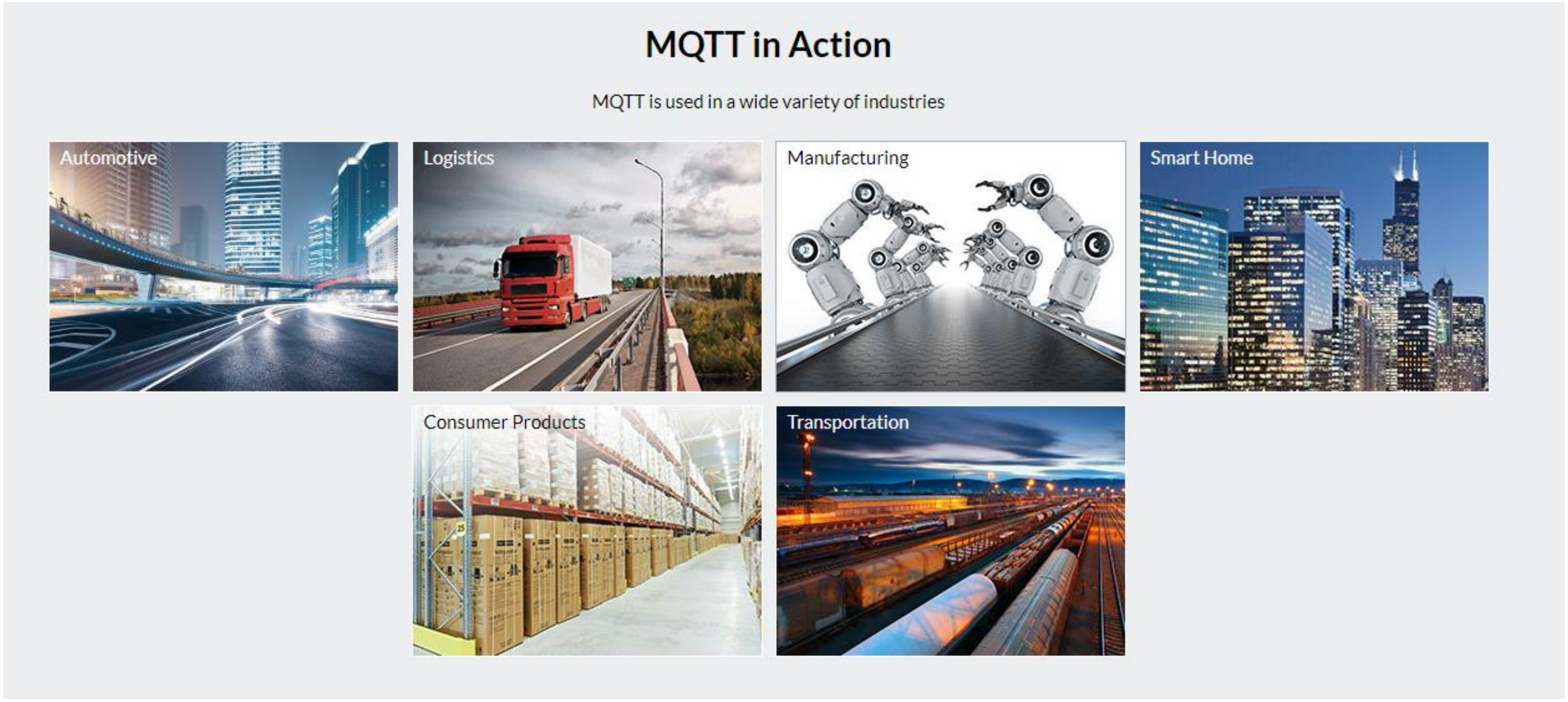
Prática 7

MQTT

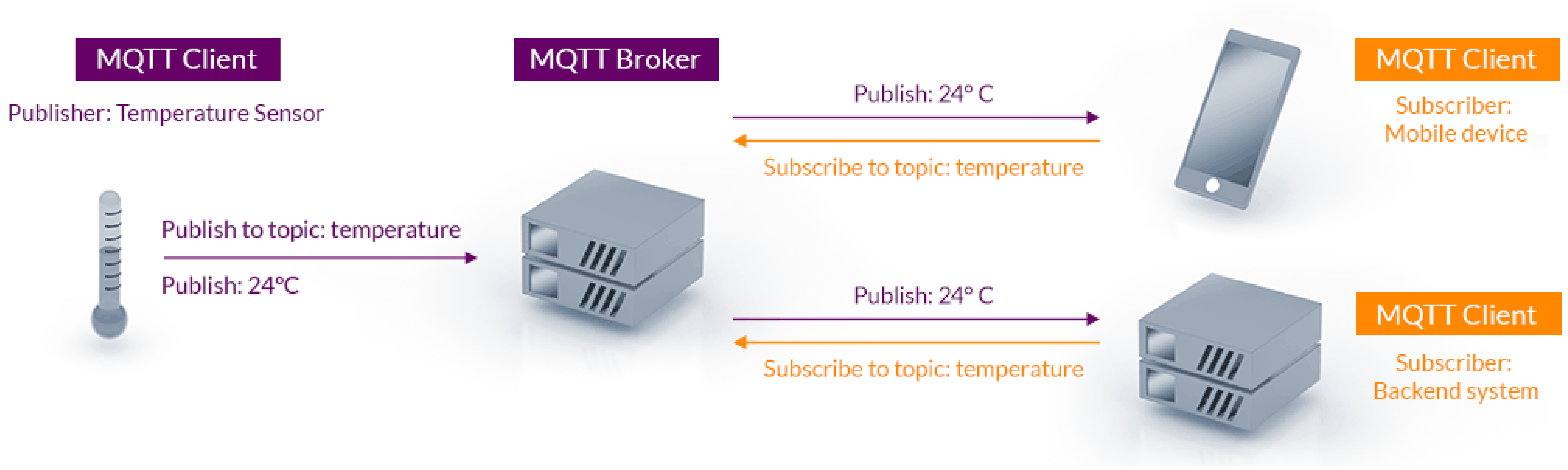
MQTT


MQTT in Action

MQTT is used in a wide variety of industries

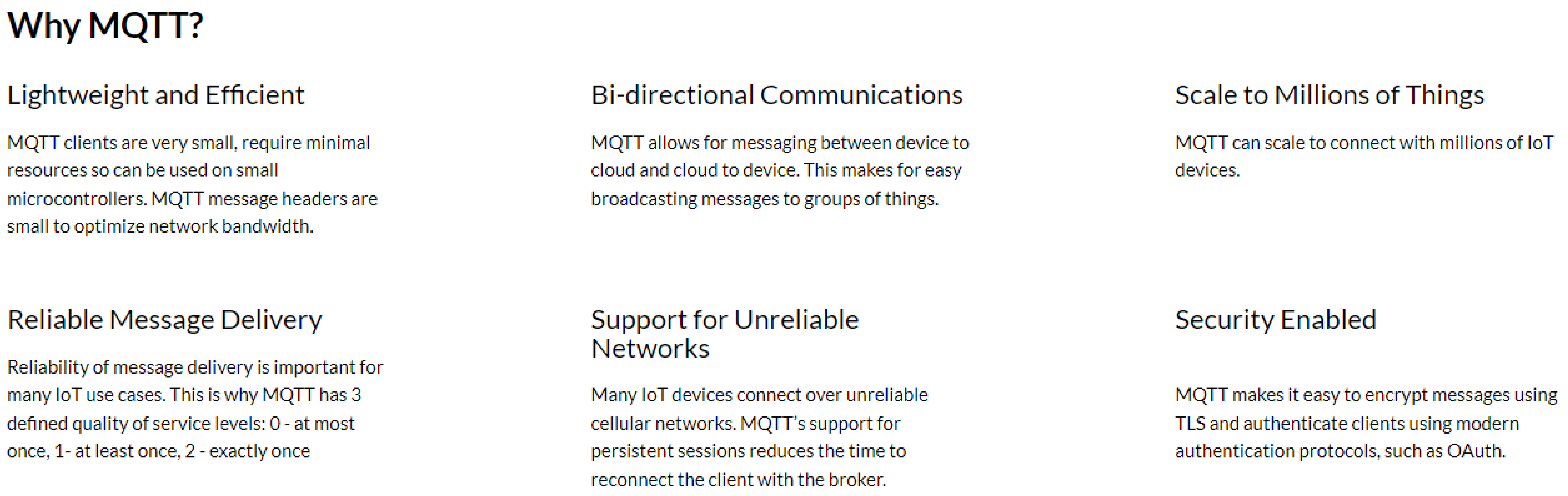
https://mqtt.org/

MQTT


MQTT Client
Publisher: Temperature Sensor
Publish to topic: temperature
Publish: 24°C
MQTT Broker
Publish: 24° C
Subscribe to topic: temperature
Publish: 24° C
Subscribe to topic: temperature
MQTT Client
Subscriber: Mobile device
MQTT Client
Subscriber: Backend system

https://mqtt.org/

MQTT



Why MQTT?

Lightweight and Efficient

MQTT clients are very small, require minimal resources so can be used on small microcontrollers. MQTT message headers are small to optimize network bandwidth.

Bi-directional Communications

MQTT allows for messaging between device to cloud and cloud to device. This makes for easy broadcasting messages to groups of things.

Scale to Millions of Things

MQTT can scale to connect with millions of IoT devices.

Reliable Message Delivery

Reliability of message delivery is important for many IoT use cases. This is why MQTT has 3 defined quality of service levels: 0 - at most once, 1- at least once, 2 - exactly once

Support for Unreliable Networks

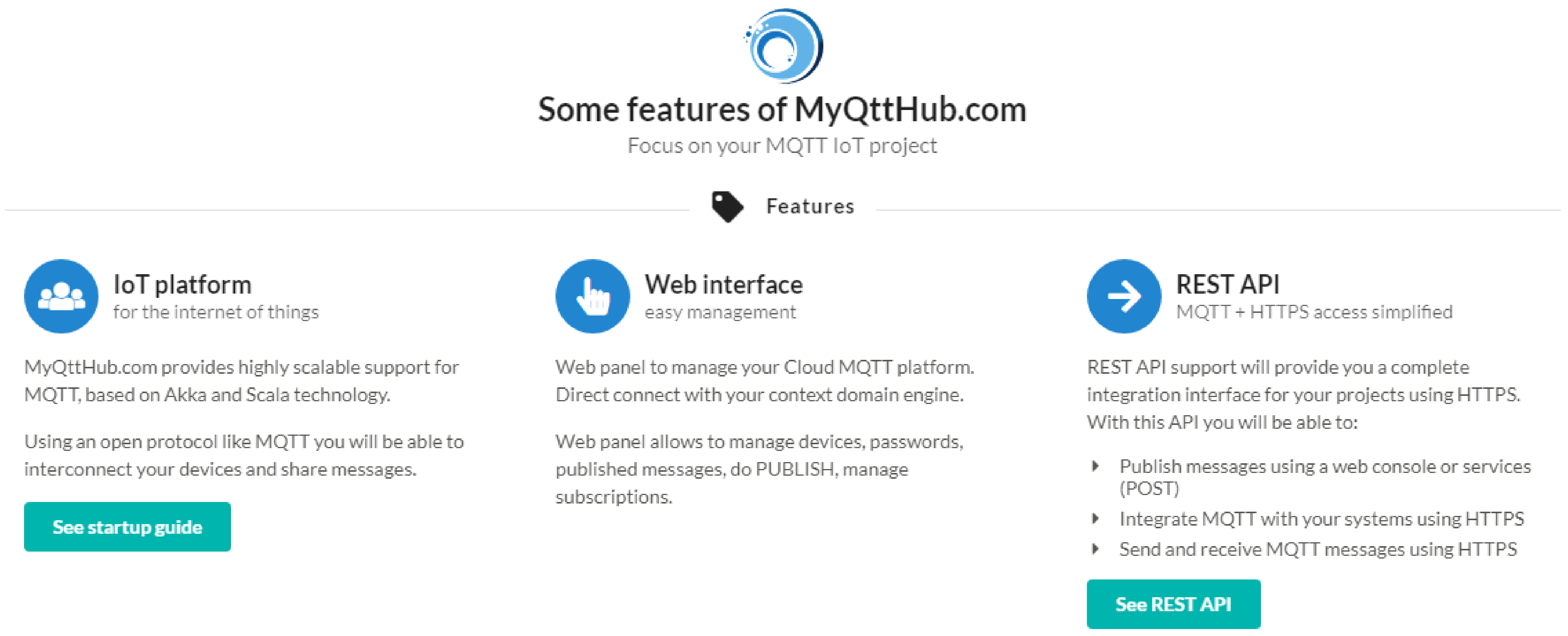
Many IoT devices connect over unreliable cellular networks. MQTT's support for persistent sessions reduces the time to reconnect the client with the broker.

Security Enabled

MQTT makes it easy to encrypt messages using TLS and authenticate clients using modern authentication protocols, such as OAuth.

https://mqtt.org/

Broker: MyQttHub.com



Some features of MyQttHub.com

Focus on your MQTT IoT project

Features

IoT platform
for the internet of things

MyQttHub.com provides highly scalable support for MQTT, based on Akka and Scala technology.

Using an open protocol like MQTT you will be able to interconnect your devices and share messages.

See startup guide

Web interface
easy management

Web panel to manage your Cloud MQTT platform. Direct connect with your context domain engine.

Web panel allows to manage devices, passwords, published messages, do PUBLISH, manage subscriptions.

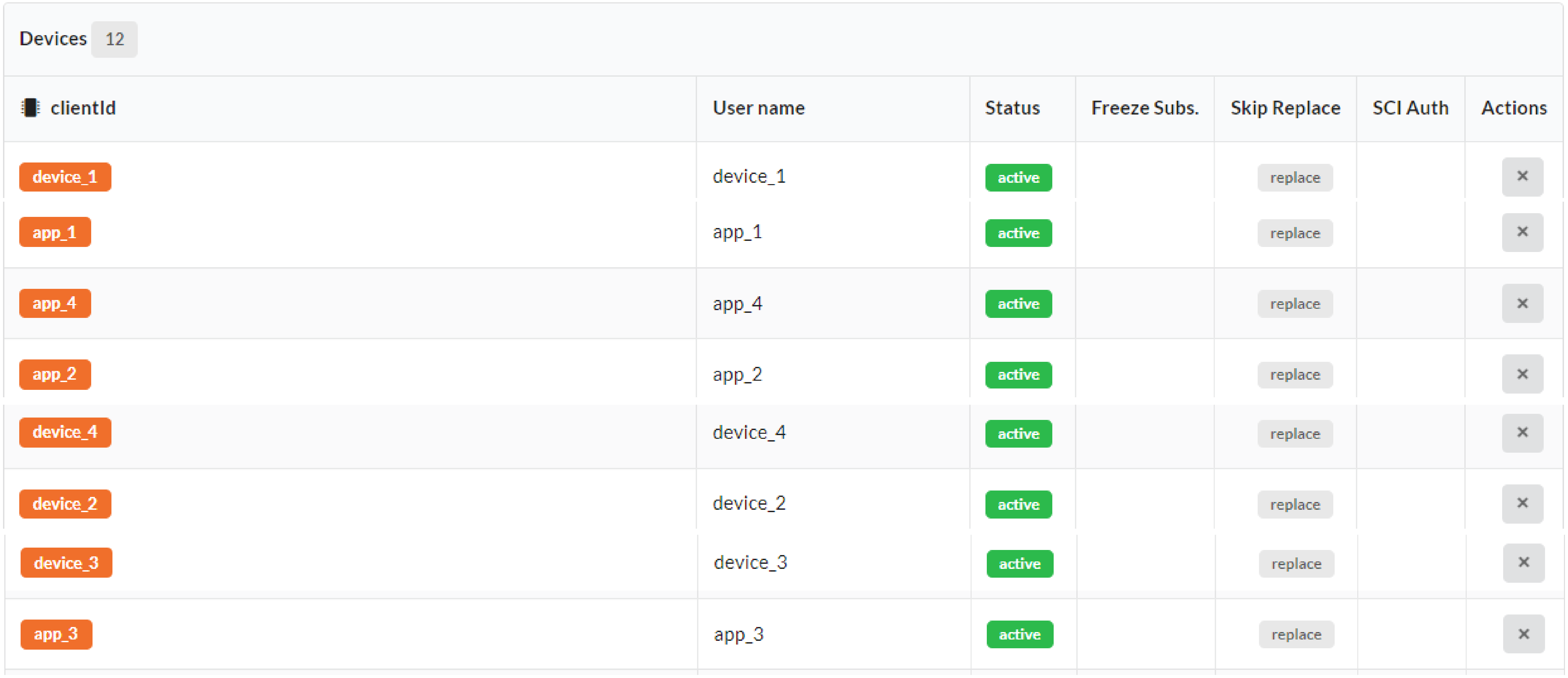
REST API
MQTT + HTTPS access simplified

REST API support will provide you a complete integration interface for your projects using HTTPS. With this API you will be able to:

- Publish messages using a web console or services (POST)
- Integrate MQTT with your systems using HTTPS
- Send and receive MQTT messages using HTTPS

See REST API

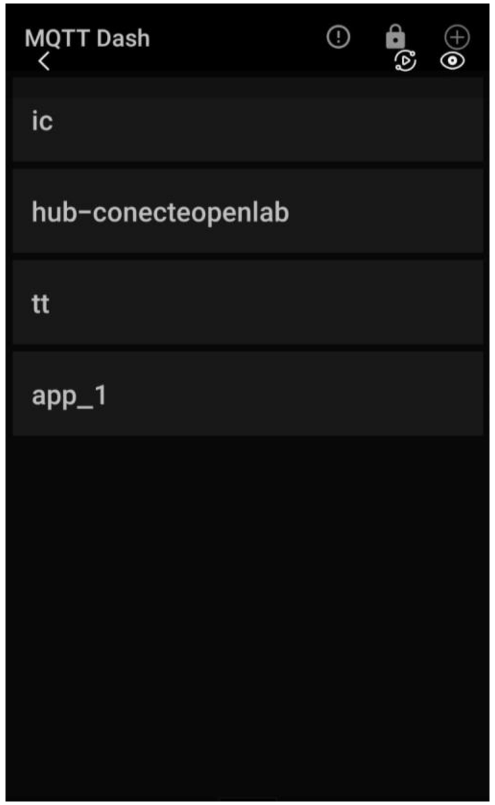
Broker: MyQttHub.com



Devices 12

clientId	User name	Status	Freeze Subs.	Skip Replace	SCI Auth	Actions
device_1	device_1	active		replace		×
app_1	app_1	active		replace		×
app_4	app_4	active		replace		×
app_2	app_2	active		replace		×
device_4	device_4	active		replace		×
device_2	device_2	active		replace		×
device_3	device_3	active		replace		×
app_3	app_3	active		replace		×

App: MQTT Dashboard

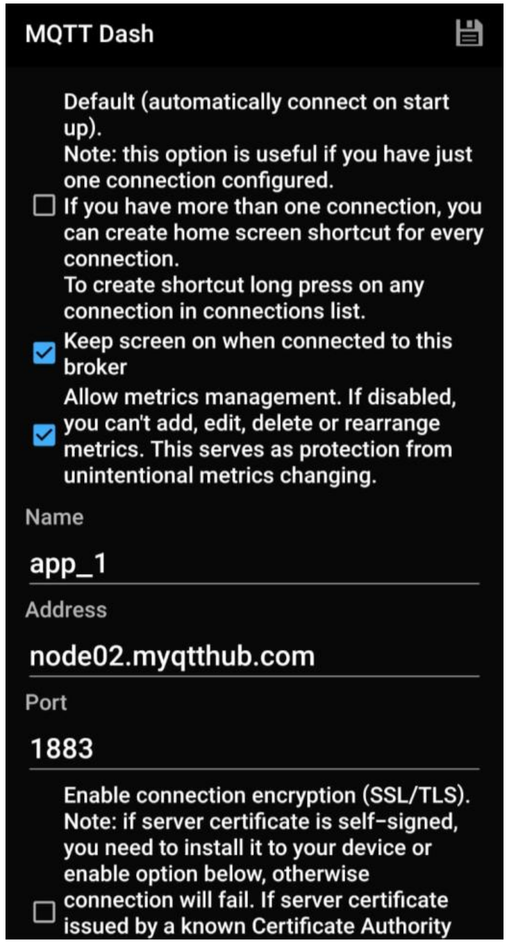

MQTT Dash

ic

hub-conecteopenlab

tt

app_1


MQTT Dash

☐ Default (automatically connect on start up).
Note: this option is useful if you have just one connection configured.
If you have more than one connection, you can create home screen shortcut for every connection.
To create shortcut long press on any connection in connections list.

☑ Keep screen on when connected to this broker

☑ Allow metrics management. If disabled, you can't add, edit, delete or rearrange metrics. This serves as protection from unintentional metrics changing.

Name

app_1

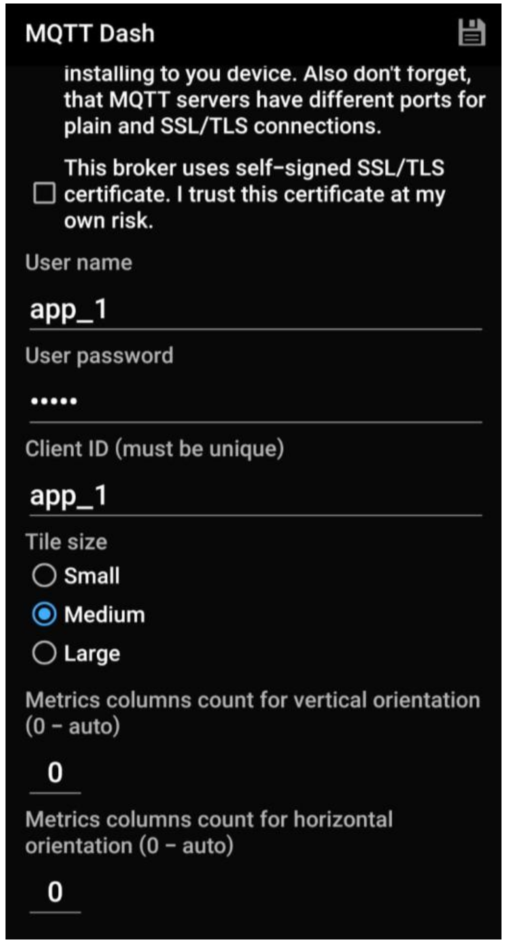
Address

node02.myqtthub.com

Port

1883

☐ Enable connection encryption (SSL/TLS).
Note: if server certificate is self-signed, you need to install it to your device or enable option below, otherwise connection will fail. If server certificate issued by a known Certificate Authority


MQTT Dash

installing to you device. Also don't forget, that MQTT servers have different ports for plain and SSL/TLS connections.

☐ This broker uses self-signed SSL/TLS certificate. I trust this certificate at my own risk.

User name

app_1

User password

•••••

Client ID (must be unique)

app_1

Tile size

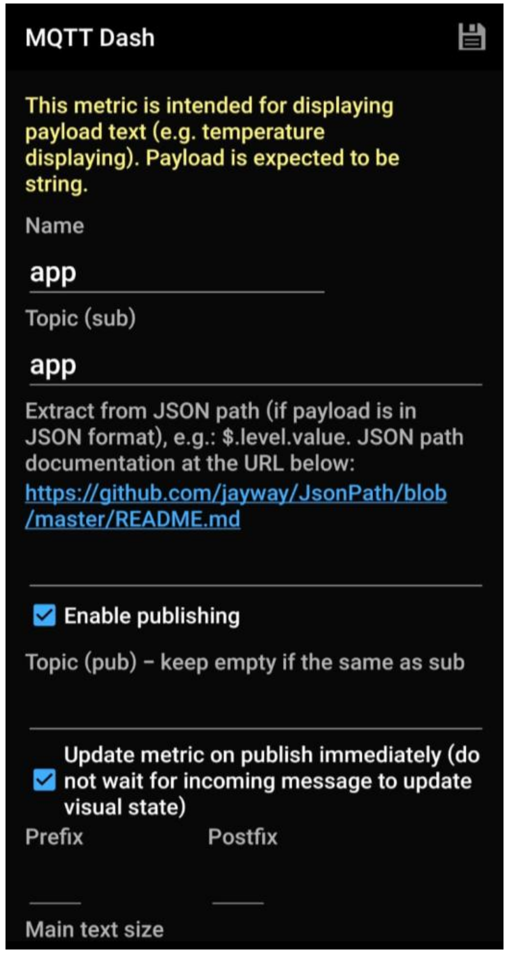
○ Small
● Medium
○ Large

Metrics columns count for vertical orientation (0 - auto)

0

Metrics columns count for horizontal orientation (0 - auto)

0


MQTT Dash

This metric is intended for displaying payload text (e.g. temperature displaying). Payload is expected to be string.

Name

app

Topic (sub)

app

Extract from JSON path (if payload is in JSON format), e.g.: $.level.value. JSON path documentation at the URL below:
https://github.com/jayway/JsonPath/blob/master/README.md

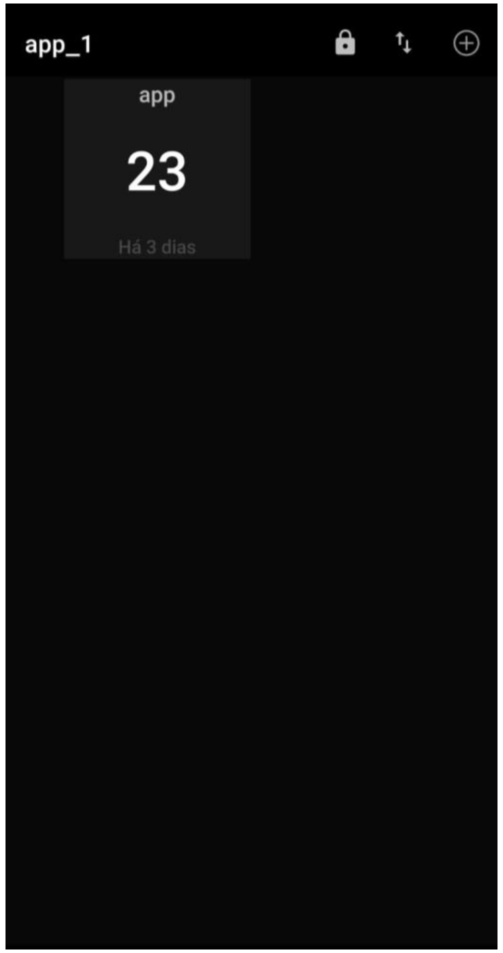
☑ Enable publishing

Topic (pub) - keep empty if the same as sub

☑ Update metric on publish immediately (do not wait for incoming message to update visual state)

Prefix Postfix

Main text size


app_1

app

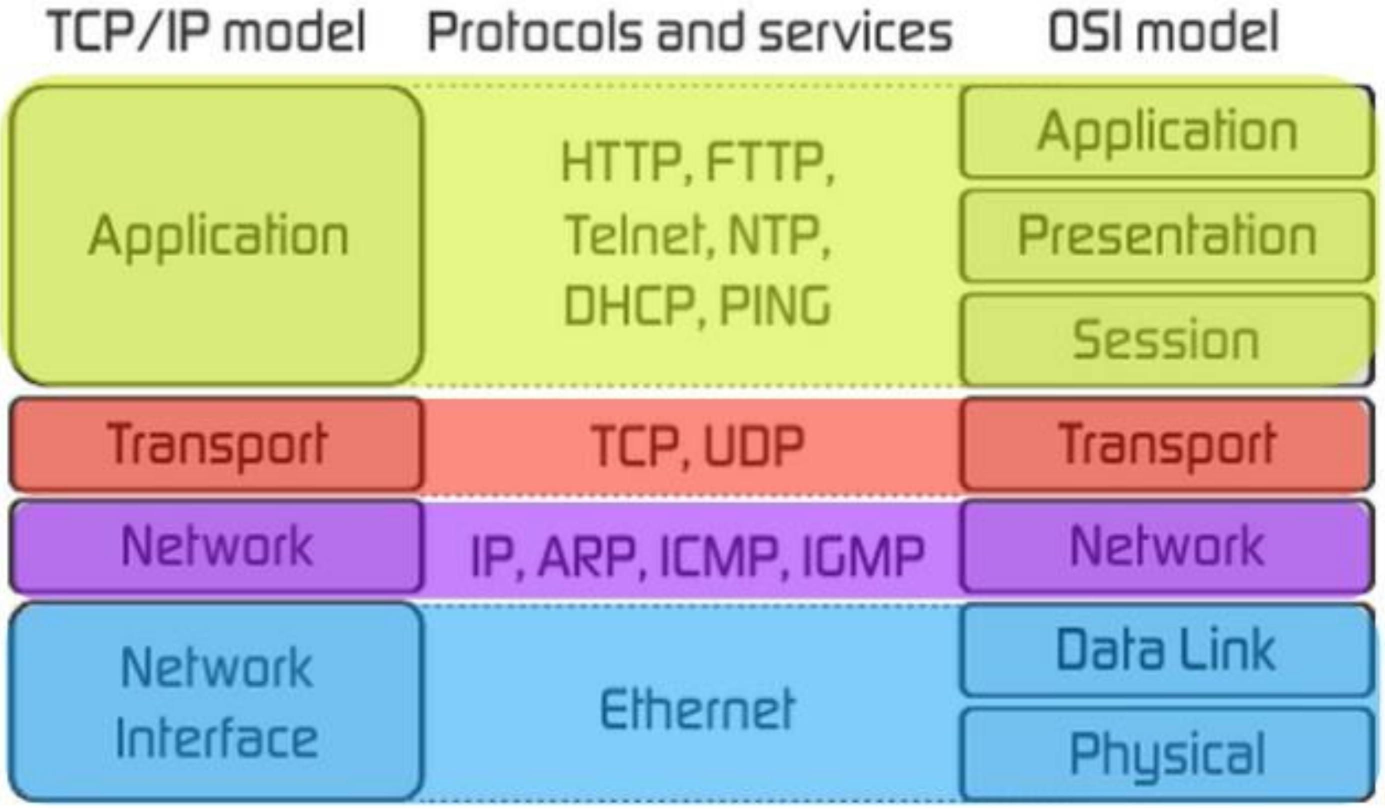
23

Há 3 dias

ESP-MQTT

- Supports MQTT over TCP, SSL with mbed tls, MQTT over Websocket, MQTT over Websocket Secure.
- Easy to setup with URI
- Multiple instances (Multiple clients in one application)
- Support subscribing, publishing, authentication, last will messages, keep alive pings and all 3 QoS levels (it should be a fully functional client).

TCP/IP



TCP/IP model	Protocols and services	OSI model
Application	HTTP, FTTP, Telnet, NTP, DHCP, PING	Application
		Presentation
		Session
Transport	TCP, UDP	Transport
Network	IP, ARP, ICMP, IGMP	Network
Network Interface	Ethernet	Data Link
		Physical

Establishing Wi-Fi or Ethernet Connection

About the example_connect() Function

- Protocols examples use a simple helper function, example_connect(), to establish Wi-Fi and/or Ethernet connection.
- It has a very simple behavior: block until connection is established and IP address is obtained, then return.
- This function is used to reduce the amount of boilerplate and to keep the example code focused on the protocol or library being demonstrated.
- The simple example_connect() function does not handle timeouts, does not gracefully handle various error conditions, and is only suited for use in examples.
- When developing real applications, this helper function needs to be replaced with full Wi-Fi / Ethernet connection handling code. Such code can be found in **examples/wifi/getting_started/** and examples/ethernet/basic/ examples.

Exemplo

```
nvs_flash_init();

esp_netif_init();

esp_event_loop_create_default();

example_connect();
```

Establishing Wi-Fi or Ethernet Connection

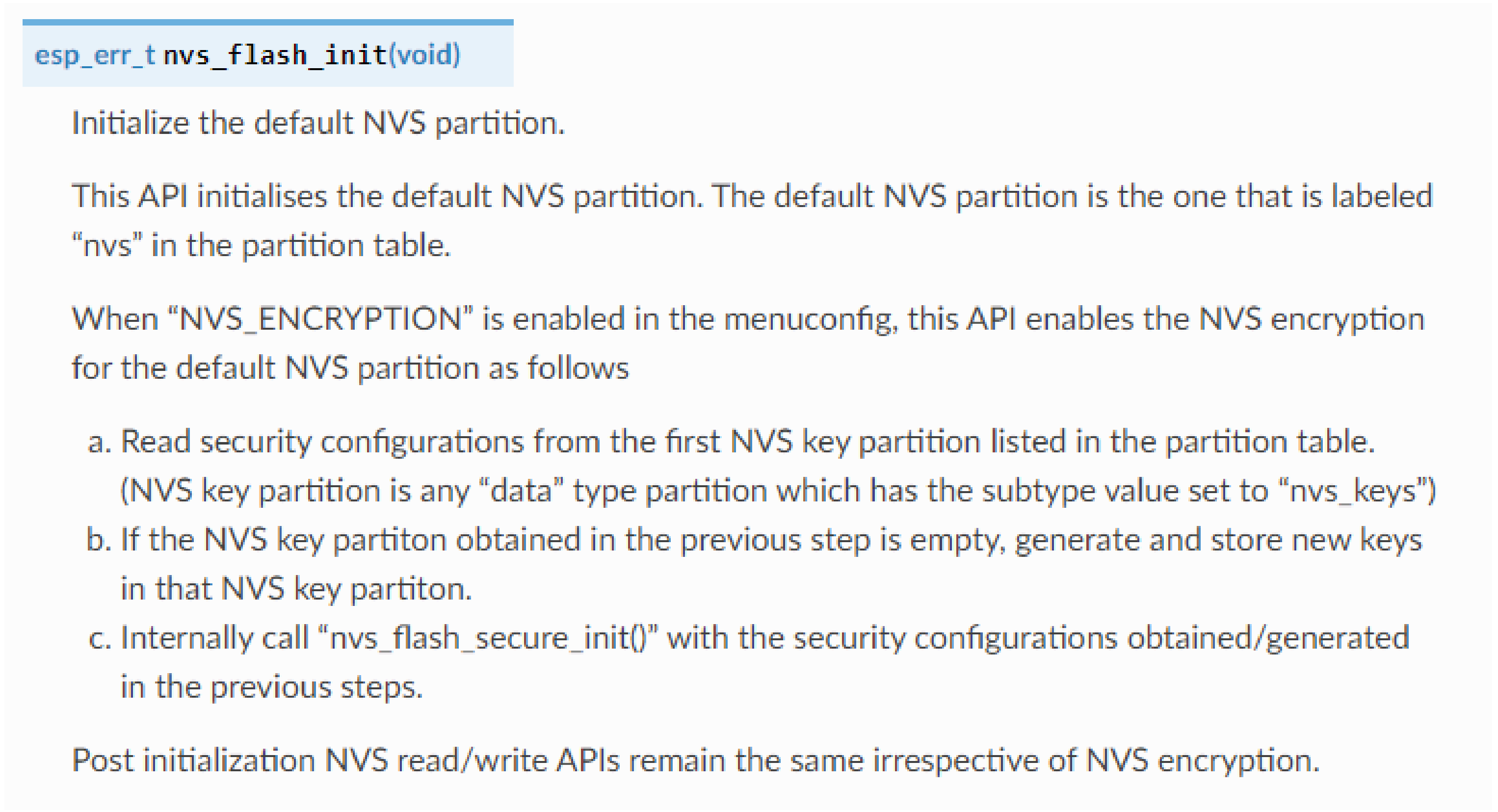
Configuring the Example

- To configure the example to use Wi-Fi, Ethernet or both connections, open the project configuration menu (**idf.py menuconfig**) and navigate to "**Example Connection Configuration**" menu. Select either "**WiFi**" or "Ethernet" or both in the "Connect using" choice.

- When connecting using Wi-Fi, enter SSID and password of your Wi-Fi access point into the corresponding fields.

Non-volatile Storage Library

Non-volatile storage (NVS) library is designed to store key-value pairs in flash



`esp_err_t nvs_flash_init(void)`

Initialize the default NVS partition.

This API initialises the default NVS partition. The default NVS partition is the one that is labeled "nvs" in the partition table.

When "NVS_ENCRYPTION" is enabled in the menuconfig, this API enables the NVS encryption for the default NVS partition as follows

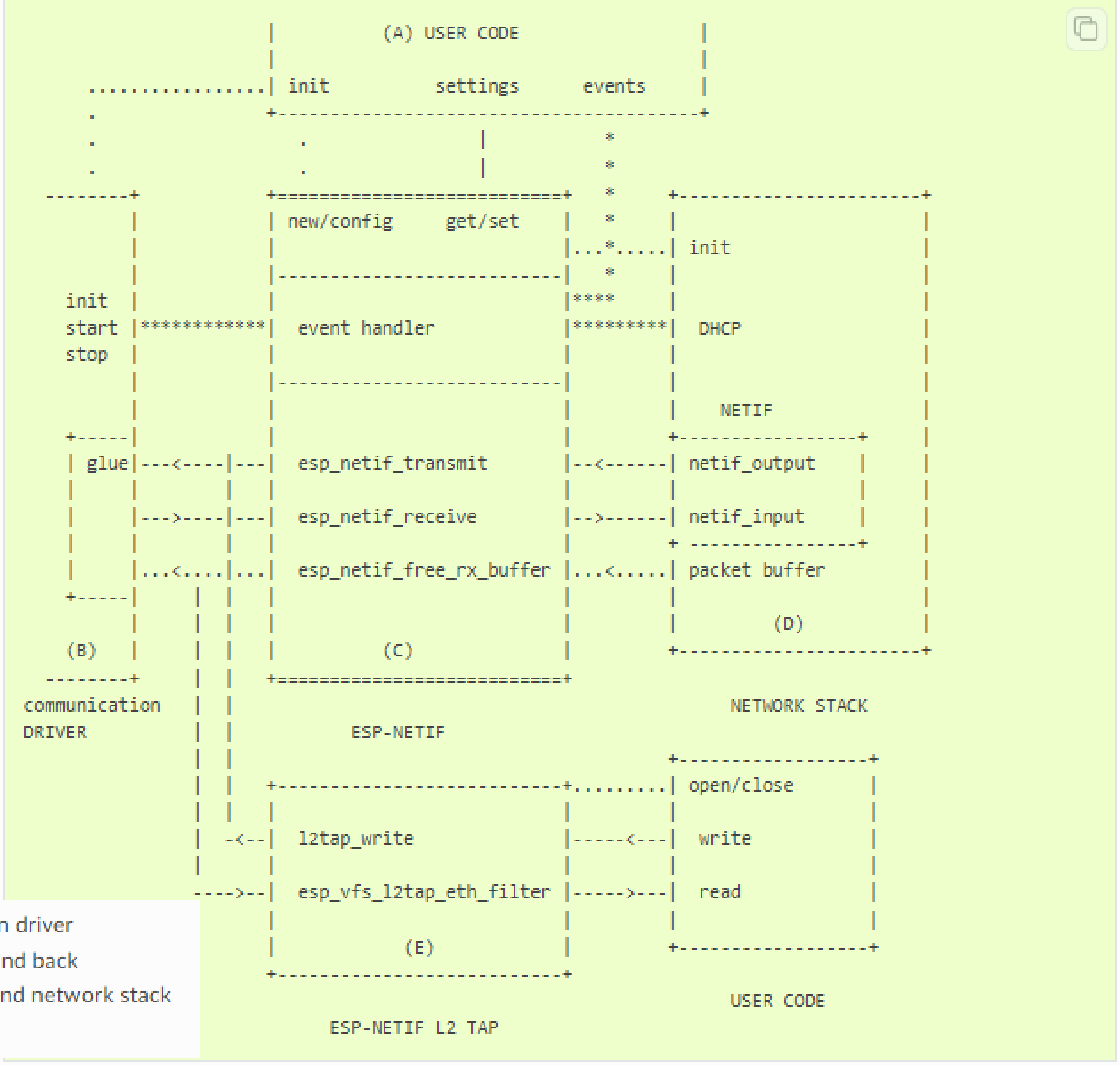
a. Read security configurations from the first NVS key partition listed in the partition table. (NVS key partition is any "data" type partition which has the subtype value set to "nvs_keys")
b. If the NVS key partiton obtained in the previous step is empty, generate and store new keys in that NVS key partiton.
c. Internally call "nvs_flash_secure_init()" with the security configurations obtained/generated in the previous steps.

Post initialization NVS read/write APIs remain the same irrespective of NVS encryption.

Netif

- It provides an abstraction layer for the application on top of the TCP/IP stack. This will allow applications to choose between IP stacks in the future.

- The APIs it provides are thread safe, even if the underlying TCP/IP stack APIs are not.



```
                       |          (A) USER CODE                 |
                       |                                        |
      .................| init          settings      events     |
      .                +----------------------------------------+
      .                   .                |           *
      .                   .                |           *
  --------+            +===========================+   *     +-----------------------+
          |            | new/config     get/set    |   *     |                       |
          |            |                           |...*.....| init                  |
          |            |---------------------------|   *     |                       |
    init  |            |                           |****     |                       |
    start |************|  event handler            |*********|  DHCP                 |
    stop  |            |                           |         |                       |
          |            |---------------------------|         |                       |
          |            |                           |         |    NETIF              |
    +-----|            |                           |         +-----------------+     |
    | glue|---<----|---|  esp_netif_transmit       |--<------| netif_output    |     |
    |     |        |   |                           |         |                 |     |
    |     |--->----|---|  esp_netif_receive        |-->------| netif_input     |     |
    |     |        |   |                           |         + ----------------+     |
    |     |...<....|...|  esp_netif_free_rx_buffer |...<.....| packet buffer         |
    +-----|     |  |   |                           |         |                       |
          |     |  |   |                           |         |          (D)          |
    (B)   |     |  |   |           (C)             |         +-----------------------+
  --------+     |  |   +===========================+
communication   |  |                                               NETWORK STACK
DRIVER          |  |           ESP-NETIF
                |  |                                         +------------------+
                |  |   +---------------------------+.........| open/close       |
                |  |   |                           |         |                  |
                |  -<--|  l2tap_write              |-----<---|  write           |
                |      |                           |         |                  |
                ---->--|  esp_vfs_l2tap_eth_filter |----->---|  read            |
                       |                           |         |                  |
                       |             (E)           |         +------------------+
                       +---------------------------+
                                                                   USER CODE
                             ESP-NETIF L2 TAP
```

....	Initialization line from user code to ESP-NETIF and communication driver
--<--->--	Data packets going from communication media to TCP/IP stack and back
********	Events aggregated in ESP-NETIF propagates to driver, user code and network stack
|	User settings and runtime configuration

Event Loop Library

Default Event Loop

The default event loop is a special type of loop used for system events (Wi-Fi events, for example). The handle for this loop is hidden from the user. The creation, deletion, handler registration/unregistration and posting of events is done through a variant of the APIs for user event loops.

By default mqtt client uses event loop library to post related mqtt events (connected, subscribed, published, etc.)

MQTT_EVENT_BEFORE_CONNECT: The client is initialized and about to start connecting to the broker.

MQTT_EVENT_CONNECTED: The client has successfully established a connection to the broker. The client is now ready to send and receive data.

MQTT_EVENT_DISCONNECTED: The client has aborted the connection due to being unable to read or write data, e.g. because the server is unavailable.

MQTT_EVENT_SUBSCRIBED: The broker has acknowledged the client's subscribe request. The event data will contain the message ID of the subscribe message.

MQTT_EVENT_UNSUBSCRIBED: The broker has acknowledged the client's unsubscribe request. The event data will contain the message ID of the unsubscribe message.

MQTT_EVENT_PUBLISHED: The broker has acknowledged the client's publish message. This will only be posted for Quality of Service level 1 and 2, as level 0 does not use acknowledgements. The event data will contain the message ID of the publish message.

MQTT_EVENT_DATA: The client has received a publish message. The event data contains: message ID, name of the topic it was published to, received data and its length. For data that exceeds the internal buffer multiple MQTT_EVENT_DATA will be posted and current_data_offset and total_data_len from event data updated to keep track of the fragmented message.

MQTT_EVENT_ERROR: The client has encountered an error. esp_mqtt_error_type_t from error_handle in the event data can be used to further determine the type of the error. The type of error will determine which parts of the error_handle struct is filled.

MQTT - example

```
esp_mqtt_client_config_t mqtt_cfg = {
        .uri = "mqtt://device_x:device_x@node02.myqtthub.com:1883", //
CONFIG_BROKER_URL,
        .client_id = "device_x",                                  //
    };


esp_mqtt_client_handle_t client = esp_mqtt_client_init(&mqtt_cfg);
    /* The last argument may be used to pass data to the event handler, in this example
mqtt_event_handler */
    esp_mqtt_client_register_event(client, ESP_EVENT_ANY_ID, mqtt_event_handler, NULL);
    esp_mqtt_client_start(client);



static void mqtt_event_handler(void *handler_args, esp_event_base_t base, int32_t
event_id, void *event_data)
{
…
}
```

MQTT – Event - examples

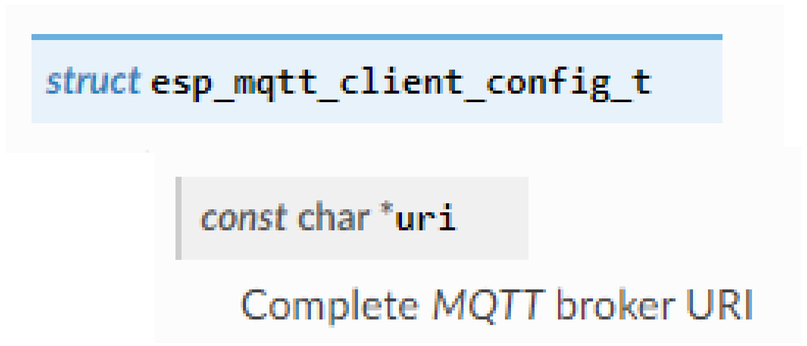
```
case MQTT_EVENT_CONNECTED:
        ESP_LOGI(TAG, "MQTT_EVENT_CONNECTED");
        msg_id = esp_mqtt_client_publish(client, "/topic/qos1", "data_3", 0, 1, 0);
        ESP_LOGI(TAG, "sent publish successful, msg_id=%d", msg_id);

        msg_id = esp_mqtt_client_subscribe(client, "/topic/qos0", 0);
        ESP_LOGI(TAG, "sent subscribe successful, msg_id=%d", msg_id);

        msg_id = esp_mqtt_client_subscribe(client, "/topic/qos1", 1);
        ESP_LOGI(TAG, "sent subscribe successful, msg_id=%d", msg_id);

        msg_id = esp_mqtt_client_unsubscribe(client, "/topic/qos1");
        ESP_LOGI(TAG, "sent unsubscribe successful, msg_id=%d", msg_id);
        break;
```

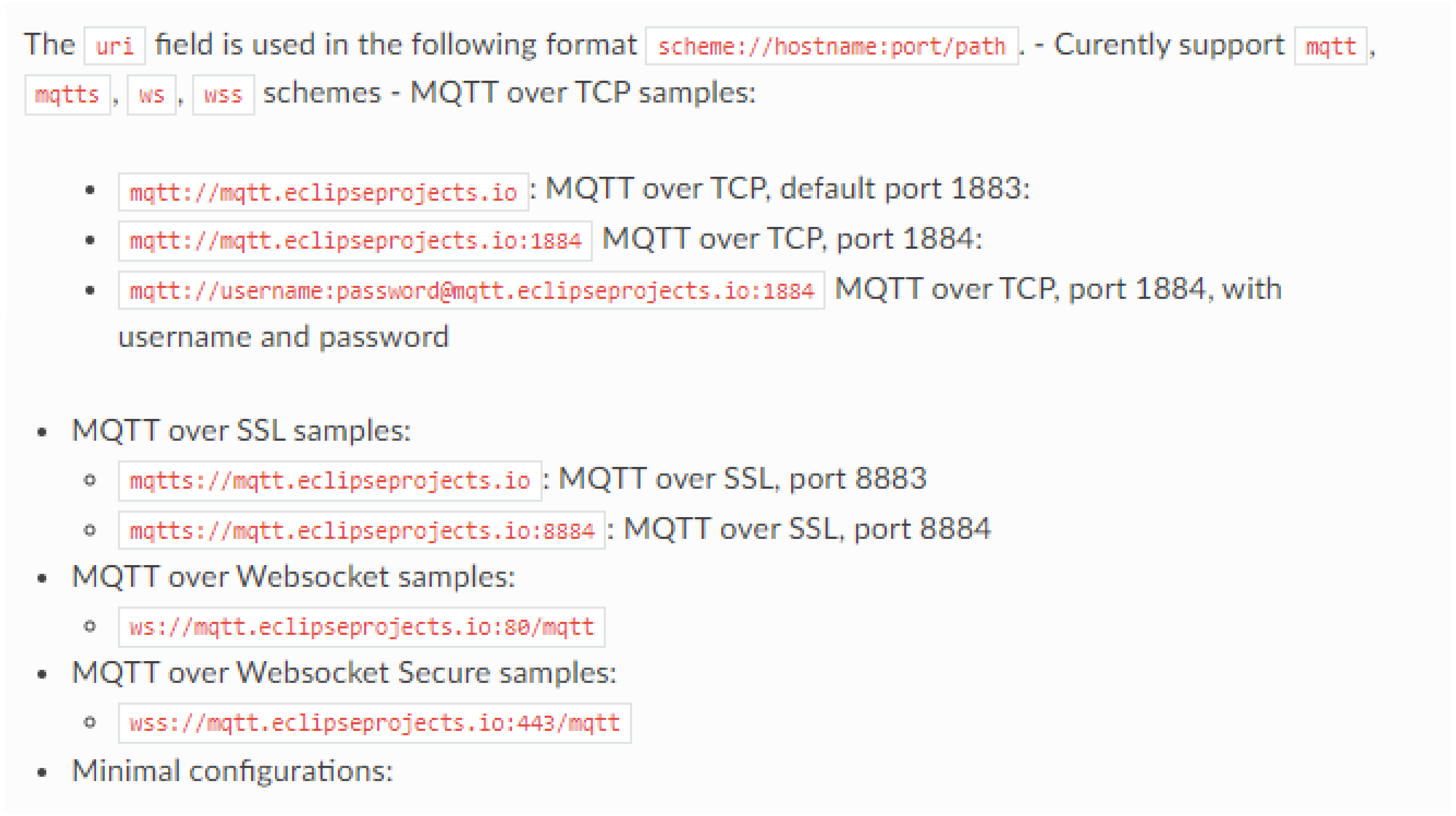
MQTT



struct **esp_mqtt_client_config_t**

const char ***uri**

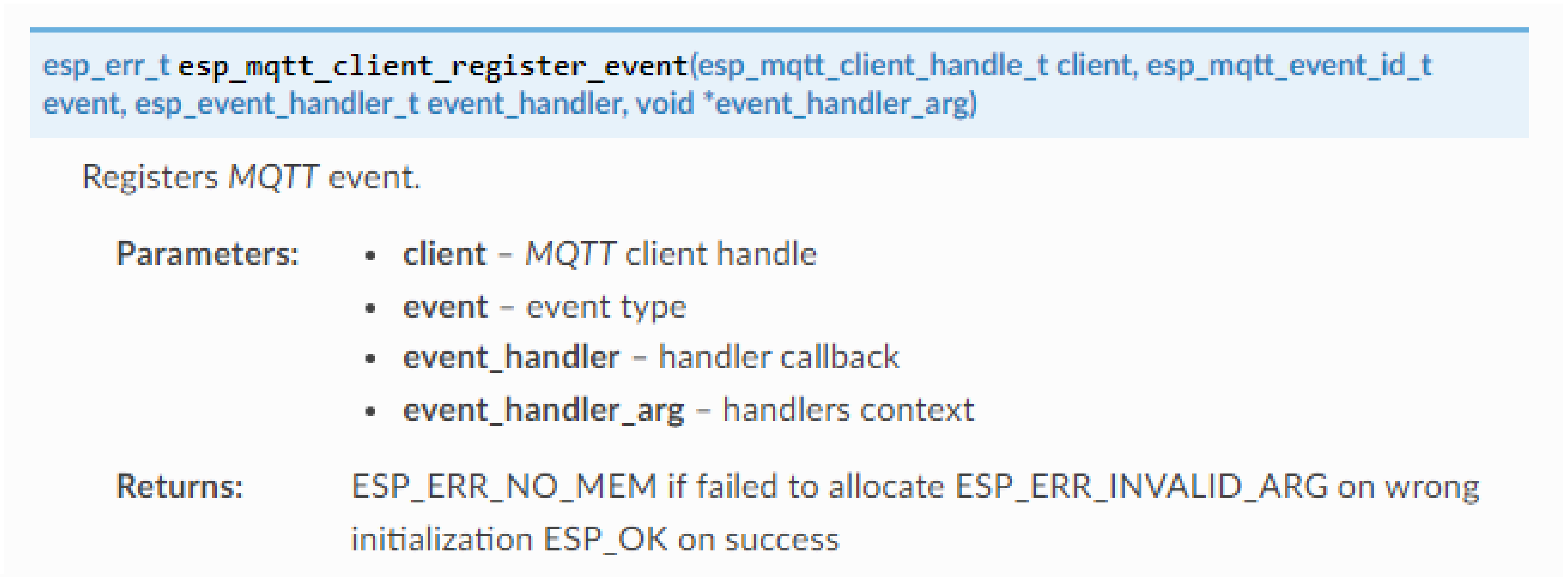
Complete *MQTT* broker URI



The `uri` field is used in the following format `scheme://hostname:port/path`. - Curently support `mqtt`, `mqtts`, `ws`, `wss` schemes - MQTT over TCP samples:

- `mqtt://mqtt.eclipseprojects.io`: MQTT over TCP, default port 1883:
- `mqtt://mqtt.eclipseprojects.io:1884` MQTT over TCP, port 1884:
- `mqtt://username:password@mqtt.eclipseprojects.io:1884` MQTT over TCP, port 1884, with username and password

- MQTT over SSL samples:
  - `mqtts://mqtt.eclipseprojects.io`: MQTT over SSL, port 8883
  - `mqtts://mqtt.eclipseprojects.io:8884`: MQTT over SSL, port 8884
- MQTT over Websocket samples:
  - `ws://mqtt.eclipseprojects.io:80/mqtt`
- MQTT over Websocket Secure samples:
  - `wss://mqtt.eclipseprojects.io:443/mqtt`
- Minimal configurations:



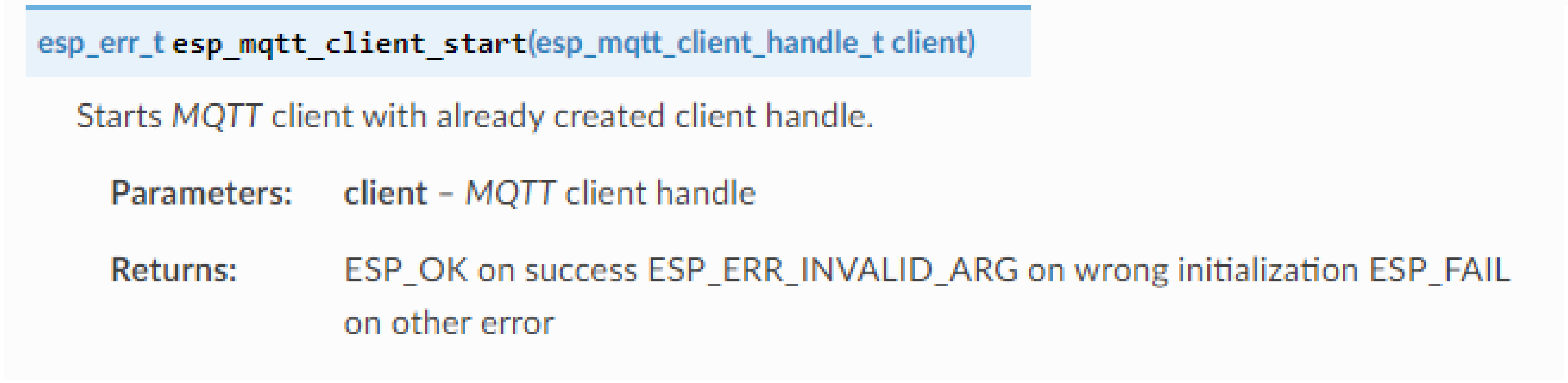
`esp_err_t esp_mqtt_client_register_event(esp_mqtt_client_handle_t client, esp_mqtt_event_id_t event, esp_event_handler_t event_handler, void *event_handler_arg)`

Registers *MQTT* event.

Parameters:

- **client** – *MQTT* client handle
- **event** – event type
- **event_handler** – handler callback
- **event_handler_arg** – handlers context

Returns: ESP_ERR_NO_MEM if failed to allocate ESP_ERR_INVALID_ARG on wrong initialization ESP_OK on success



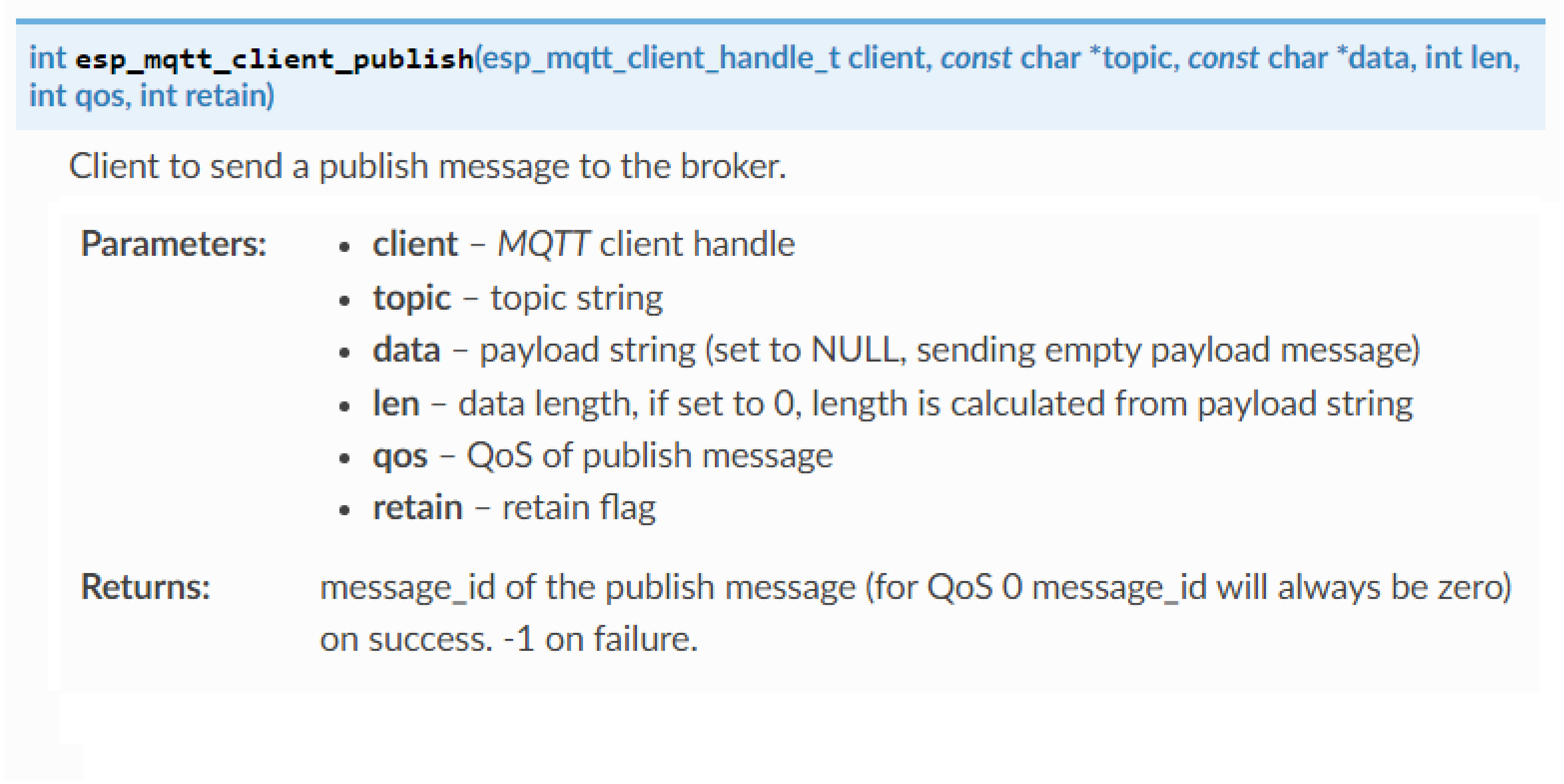
`esp_err_t esp_mqtt_client_start(esp_mqtt_client_handle_t client)`

Starts *MQTT* client with already created client handle.

Parameters: **client** – *MQTT* client handle

Returns: ESP_OK on success ESP_ERR_INVALID_ARG on wrong initialization ESP_FAIL on other error

Publish



int **esp_mqtt_client_publish**(esp_mqtt_client_handle_t client, *const* char *topic, *const* char *data, int len, int qos, int retain)

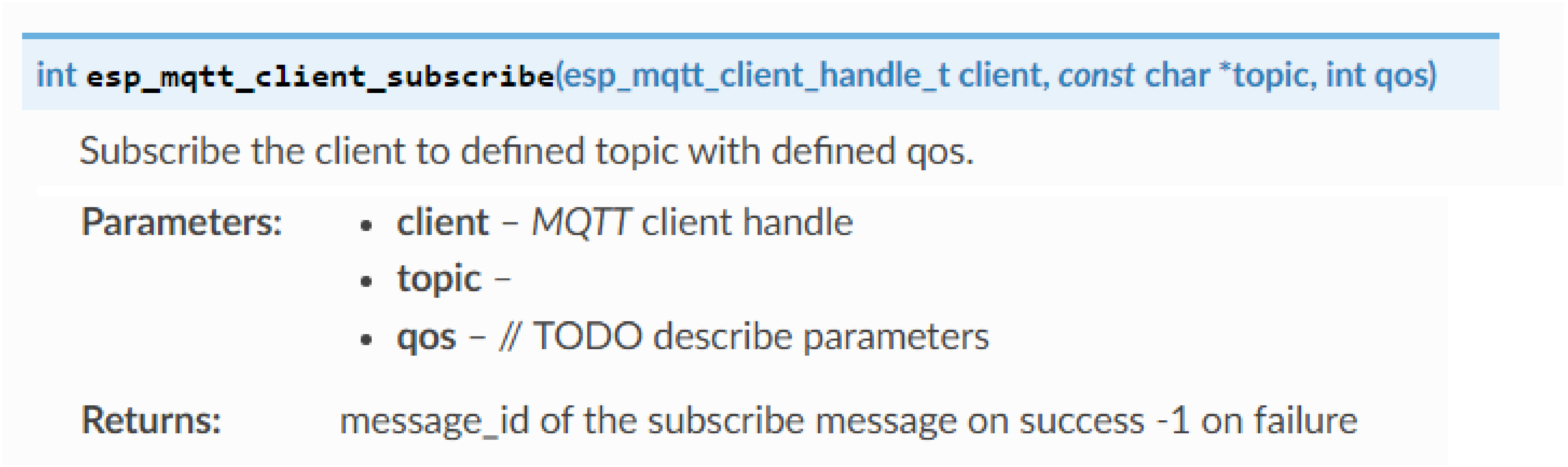
Client to send a publish message to the broker.

Parameters:

- **client** – *MQTT* client handle
- **topic** – topic string
- **data** – payload string (set to NULL, sending empty payload message)
- **len** – data length, if set to 0, length is calculated from payload string
- **qos** – QoS of publish message
- **retain** – retain flag

Returns: message_id of the publish message (for QoS 0 message_id will always be zero) on success. -1 on failure.

Subscribe



int **esp_mqtt_client_subscribe**(esp_mqtt_client_handle_t client, *const* char *topic, int qos)

Subscribe the client to defined topic with defined qos.

Parameters:
- **client** – *MQTT* client handle
- **topic** –
- **qos** – // TODO describe parameters

Returns: message_id of the subscribe message on success -1 on failure

Referências

- https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf