

# IoT Engineering

## 10: Rule-based Integration of IoT Devices

CC BY-SA, Thomas Amberg, FHNW  
(unless noted otherwise)

Slides: [tmb.gr/iot-10](https://tmb.gr/iot-10)

# Overview

These slides introduce *integration of IoT devices*.

How a sensor can trigger a (remote) actuator.

How multiple devices can be combined.

How to talk to third party products.

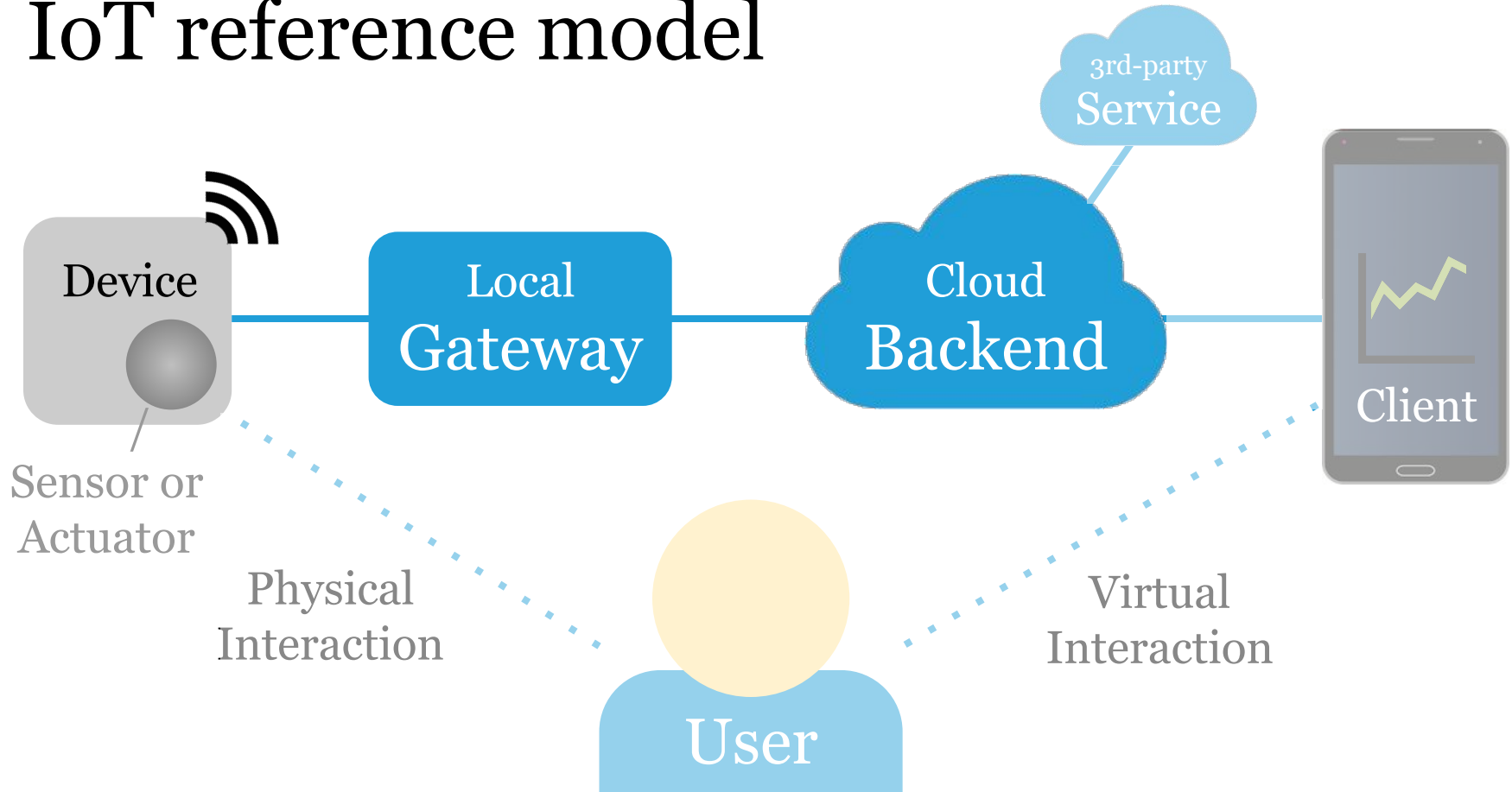
# Prerequisites

The **Raspberry Pi** with **Node-RED** will be our gateway.

We use **curl** and the **mqtt** CLI tool to emulate devices.

And the **Feather Huzzah ESP8266** as a real device.

# IoT reference model



# Remote sensing

Sensor  $\rightarrow$  Device  $\rightarrow$  Gateway  $\rightarrow$  Backend  $\rightarrow$  Client

Data is sent from a device, to a client, via a backend.

E.g. air quality sensor measurements sent to a map.

$A \rightarrow B$  means data flows from A to B.

# Remote control

Client → Backend → Gateway → Device → Actuator

Control data is sent to a device and on to an actuator.

E.g. app sends command via backend to dim a light.

Or a stormy weather service triggers a blind to go up.

Remote sensing and control can be integrated.

# Levels of control

Thin API, providing detailed access to the hardware:

PUT `https://MY_HOUSE/room/lamp?color=0xffffffff`

Simple, easy to use API, lamp chooses color settings:

PUT `https://MY_HOUSE/room/lamp?state=on`

Semantically rich API, involving multiple devices:

PUT `https://MY_HOUSE/room?scene=relax`

HTTP is an implementation detail here.

# Hands-on, 5': Where is the logic?

Who decides what a scene means in terms of color, the lamp/device, a room/gateway or the backend?

Which information is required to make a decision?

Which devices are affected by changing a scene?

Which trade-offs does placing the logic involve?



# Logic trade-offs

Logic on the backend or client, "in the cloud" — one central place to change functionality for all devices.

Logic on the gateway, "at the edge" — less latency, adapted to local topology, but local information only.

Logic on the device — works when a device is offline, but requires per device firmware update for changes.

# Rule-based integration

Control data is sent based on sensor measurements.

*Rules* describe the conditions which trigger events.

Integrating sensors & actuators of separate devices.

Integration can happen at different levels.

# Integration on the backend

Client ← Backend ← Gateway ← Device ← Sensor  
→ Gateway → Device → Actuator

Rules on backend integrate 2..n devices in n locations.

E.g. Nest thermostat, learning based on global data.

# Integration on the gateway

Client ← Backend ← Gateway ← Device ← Sensor  
→ Device → Actuator

Rules on gateway integrate 2..n devices in 1 location.

E.g. a building automation system controlling heat.

# Message brokers

A simple way to integrate devices is through a broker.

E.g. a button can publish its state (*on* or *off*) to a topic.

And a lamp device can subscribe to the button's topic.

Or a 3rd party can create the link, to keep them apart:

Subscribe to the button, publish to the lamp's topic.

# Glue code

Glue code works well to integrate devices and services.

E.g. as the 3rd party in the previous (broker) example.

Or as a bridge from a local broker to a cloud backend.

The code can inspect and transform messages.

# Node-RED

**Node-RED** enables flow-based programming for IoT.

The tool/service runs on a gateway or "in the cloud".

A Web UI allows users to connect devices & services.

Program **flows** can be exported/imported as JSON.

Modular **nodes** allow using 3rd-party functionality.

# Installing Node-RED

To install & run Node-RED on the Raspberry Pi, type:

```
$ sudo apt-get install build-essential  
$ bash <(curl -sL https://raw.githubusercontent.com/node-red/linux-installers/master/deb/  
/update-nodejs-and-nodered)
```

To enable autostart of Node-RED after reboot, type:

```
$ sudo systemctl enable nodered.service  
$ sudo reboot
```



# Securing Node-RED

To **hash and set the password** for your Node-RED user:

```
$ sudo npm install -g node-red-admin
```

```
$ node-red-admin hash-pw
```

```
$ nano .node-red/settings.js
```

```
adminAuth: { // TODO: find, uncomment this part
  type: "credentials",
  users: [{ username: "USERNAME", // TODO
    password: "PASSWORD_HASH", // TODO
    permissions: "*" } ] },
```

# Accessing Node-RED

To access Node-RED on the Raspberry Pi, either:

- Connect to the same network, then access the Node-RED Web UI at `http://RASPI_IP:1880/`
- Make 127.0.0.1 port 1880 of the Pi accessible via a relay service like [Ngrok](#), [PageKite](#) or [Yaler](#).

Node-RED : fhnw-iot-36-4c-fe

+

← → ↺

https://fhnw-iot-\*\*\*\*\*-1880.try.yaler.io/#

🔑 🔍 ☆ 🗑️ 🛠️ 👤

Node-RED

Deploy

👤 ☰

🔍 mqtt ×

Flow 1

Flow 2

+

☰

input

mqtt

output

mqtt

function

social

storage

analysis

advanced

Raspberry Pi

temp

connected

msg.payload

> 23?

{ on }

{ off }

alert

connected

humi

connected

> 42?

email

info

🔍 🛠️ ⌵

Information

Node "19438844.9115c8"

Name > 23?

Type switch

show more

Description

Node Help

Route messages based on their property values or sequence position.

Details

When a message arrives, the node will evaluate each of the defined rules and forward the message to the

# How Node-RED works

Node-RED maps inputs to outputs with functions.

Functions can aggregate, switch, transform, etc.

The basic unit of information is a message.

A message has a payload and metadata.

For details, see [Node-RED docs](#).

# Node-RED MQTT client

Use the *mqtt* node to act as a publisher or subscriber.

Subscribe to messages, publish alerts to another topic.

Subscribe to local messages, publish them to the cloud.

Act depending on the value of the message payload.

# Node-RED HTTP Web service

Make a Web service with *http* & *http-response* nodes.

Receive Webhook calls, transform the body payload.

Forward HTTP Webhook data to an MQTT broker.

Or use the *http request* node to (also) be a client.

# Hands-on, 15': Node-RED

Install Node-RED on the Raspberry Pi or your laptop.

Create a new flow or import & analyse one from [here](#).

Use the *debug* node to build your flow step-by-step.

# Integration on a 3rd party service

Service\*  $\leftarrow$  Backend<sub>1</sub>  $\leftarrow$  Gateway  $\leftarrow$  Device  $\leftarrow$  Sensor  
                     $\rightarrow$  Backend<sub>n</sub>  $\rightarrow$  Gateway  $\rightarrow$  Device  $\rightarrow$  Actuator

3rd party integrates 2..n devices/connected products.

Integration is done via product specific backend APIs.

Rules specify how a sensor input triggers actuators.

A typical example for this is IFTTT, see next slide. 24



# IFTTT

IFTTT enables the integration of devices and services.

"if this then that" *applets* connect *triggers* to *actions*.

Devices trigger *events* or are controlled by an action.

Many connected products have IFTTT integrations\*.

\*Hue, Nest, Netatmo, Oticon, Ring, Withings, ...

# IFTTT applets

End-users can **create an IFTTT applet** themselves.

Each product or service has to be *connected* once.

Connecting here means getting API permissions.

After this setup step IFTTT keeps track of events.

Get a notification in your hearing aids when the doorbell rings.

[Click for full guide](#)



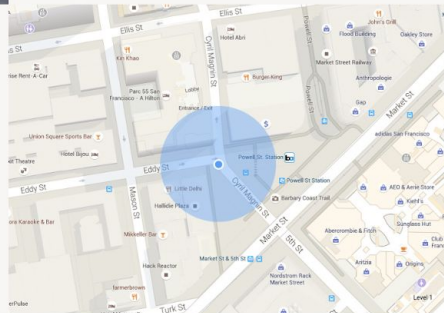
Receive a notification when your home alarm system goes off.

[Click for full guide](#)



Send a text message or email to a caregiver when the hearing aid battery is low.

[Click for full guide](#)



Have your hearing aids switch to a preferred listening program based on your GPS location.

[Click for full guide](#)

Have your hearing aids go into a special listening program when you make a voice command.

[Click for full guide](#)



# IFTTT Webhooks

**IFTTT Webhooks** is a simple way to integrate devices.

It enables triggers and actions for quick prototyping\*.

"Webhook" includes incoming and outgoing calls.

IFTTT sends or receives HTTP Web requests.

\*Real products use the **IFTTT platform** API.

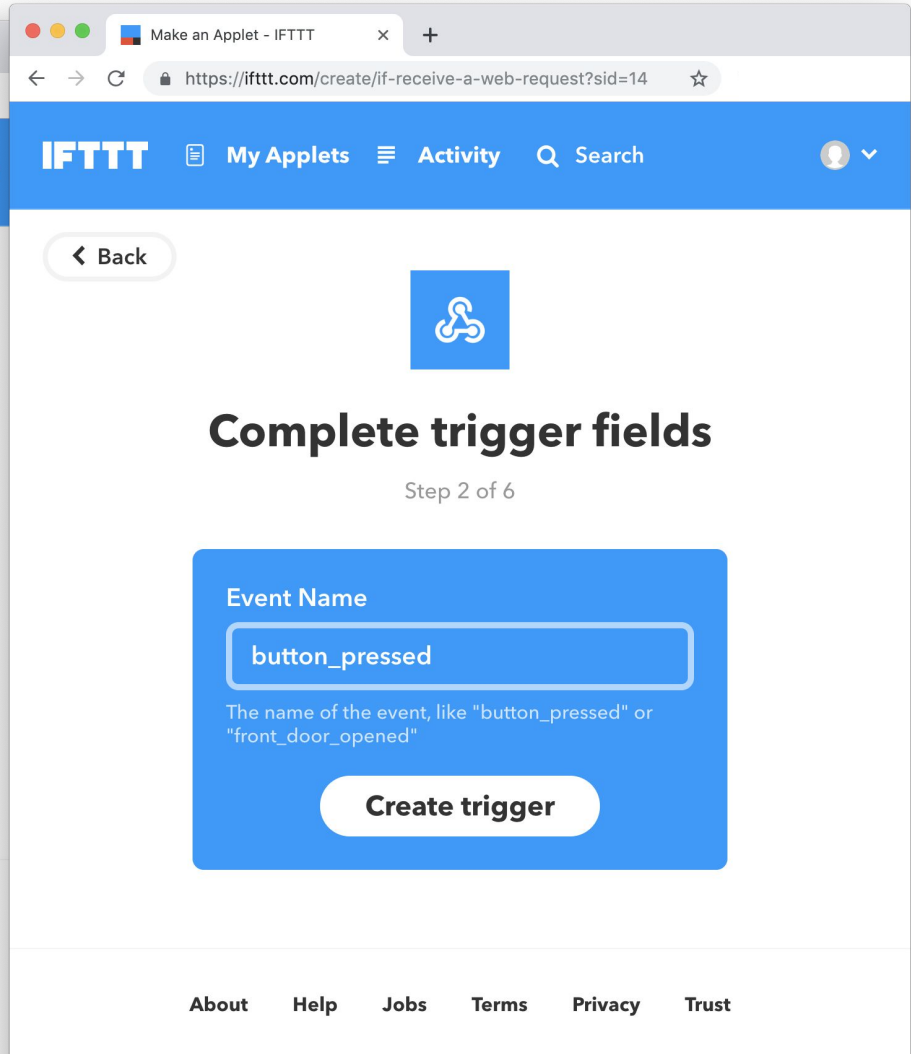
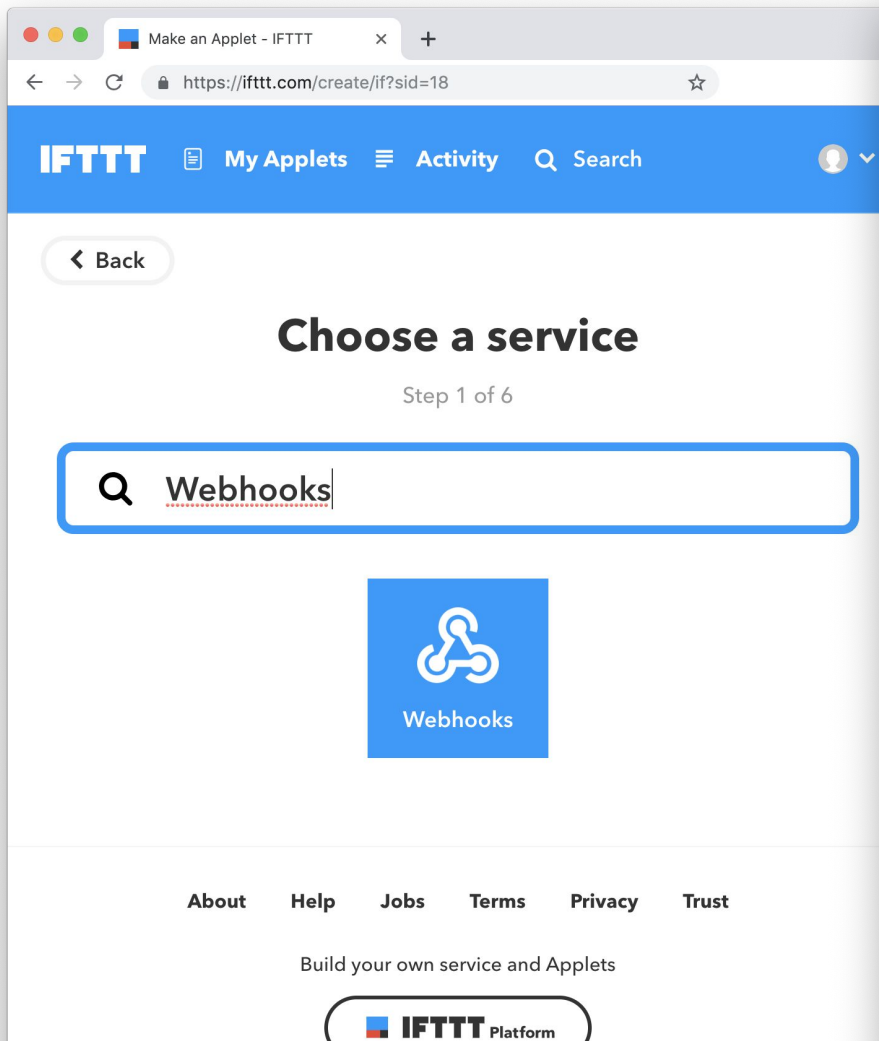
# IFTTT Webhook trigger

IFTTT can receive Web requests to *trigger* an *event*.

To get a *key*, see [IFTTT Webhooks](#) > *Documentation*.

```
$ curl -X -H "Content-Type: application/json"  
-d '{"value1": "23", "value2": "42"}'  
https://maker.ifttt.com/trigger/MY_EVENT/with/  
key/IFTTT_API_KEY
```

The POST request body and values are optional.



# Hands-on, 15': IFTTT Webhook trigger

Imagine an IFTTT Webhook enabled button device.

Create an applet to send SMS if the button is pressed.

Emulate the *button\_pressed* event using the curl tool.

Sketch the hardware and code to build such a button.

Done? Build your connected button with Arduino.

# IFTTT Webhook action

IFTTT can make a Web request as a resulting *action*.

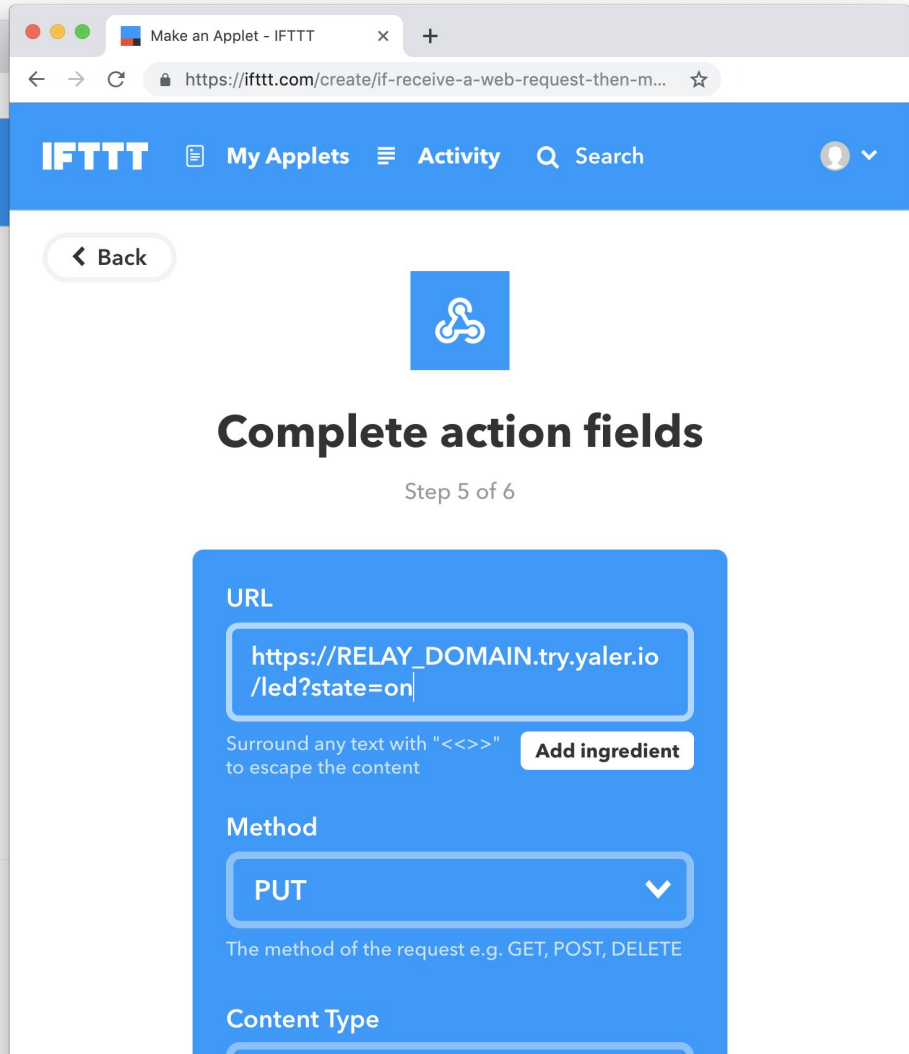
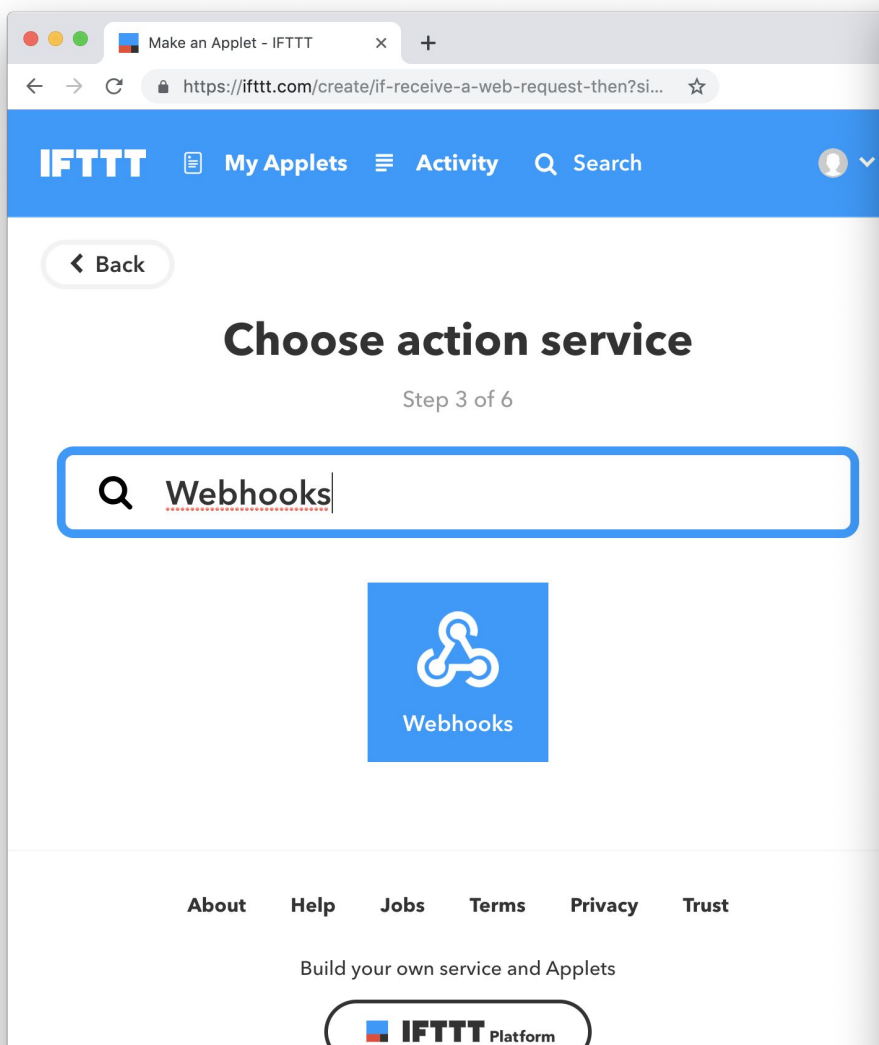
The outgoing Webhook API calls a URL you provide.

It supports PUT and POST methods, among others.

The body can be JSON, URL-encoded or plain text.

Variable *ingredients* depend on the chosen trigger.





# Hands-on, 15': IFTTT Webhook action

Create an IFTTT applet to show the [weather](#) on a [LED](#).

Design a Web API\* to set weather conditions or colors.

Where would you map weather conditions to colors?

Create a [Postb.in](#) to test the IFTTT Webhook call.

\*Which would run e.g. on the ESP8266 or a Pi.

# Summary

We've seen on which level integration can happen.

How integration is done with glue code or services.

On the gateway we used Node-RED to implement it.

In the backend we integrated products with IFTTT.

Next: Voice Control for Connected Products.

# Feedback or questions?

Write me on Teams or email

[thomas.amberg@fhnw.ch](mailto:thomas.amberg@fhnw.ch)

Thanks for your time.