Introduzione a Micropython

Da zero al Cloud

con **Juna Salviati**

Chi sono

Juna Salviati
Solutions Architect
Technical communities enthusiast!

@1littleendian
https://medium.com/@1littleendian
https://dev.to/antigones

Passare al Cloud: alcuni servizi Azure per l’IoT

Overview

Due servizi:

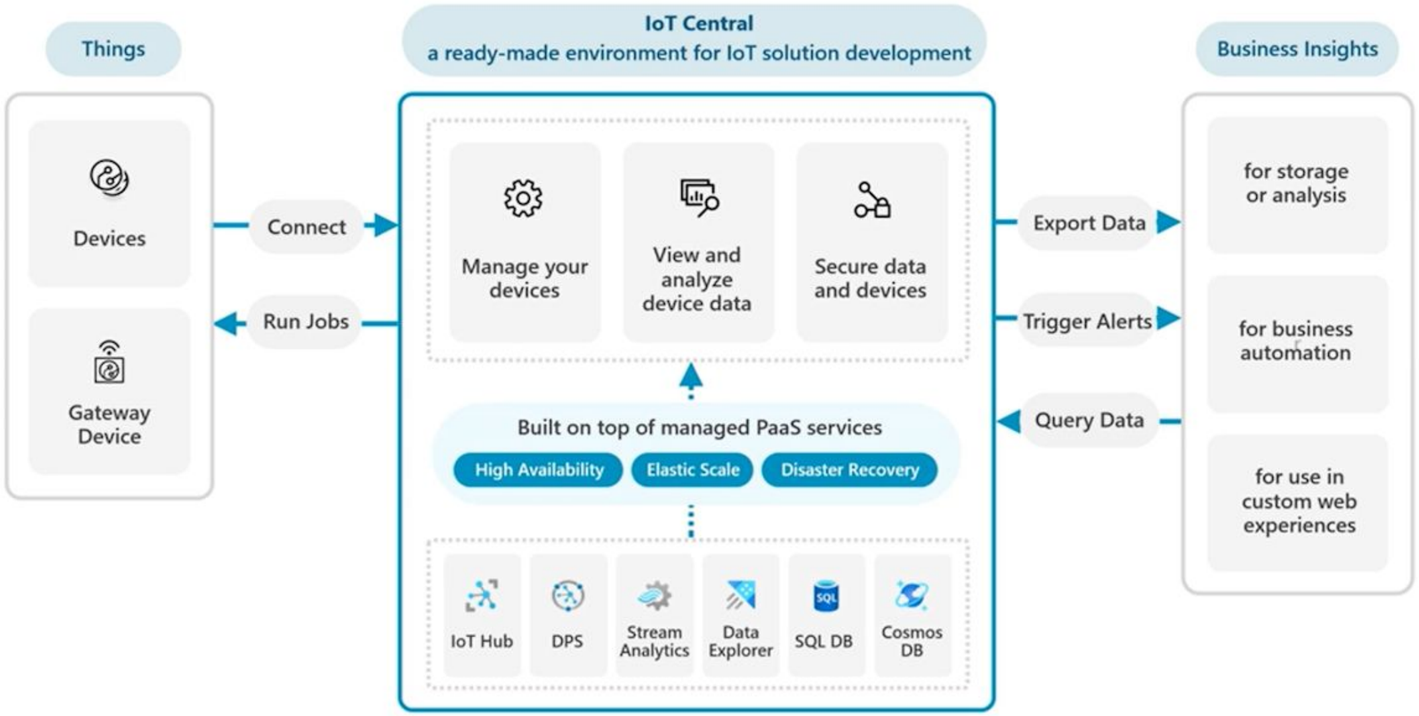
- IoTHub (PaaS)
  - un hub di messaggi device-cloud e cloud-device
  - attorno al quale costruire applicazioni personalizzate
    - es: IoTHub+Stream Analytics -> ML sui dati
  - senza alcuna interfaccia per i dati
- IoTCentral (aPaaS)
  - una soluzione “all-in-one”

Azure IoTCentral

Un servizio Cloud Azure “all-in-one” per una interazione “agevolata” con i device:

- strutture ad-hoc per l’interazione con i device
- possibilità di creare dashboard per la visualizzazione
- possibilità di creare regole sui valori letti dai device
- possibilità di esportare i dati verso altre destinazioni
- offre una REST API per l’interazione

IoTCentral - Una pluralità di servizi

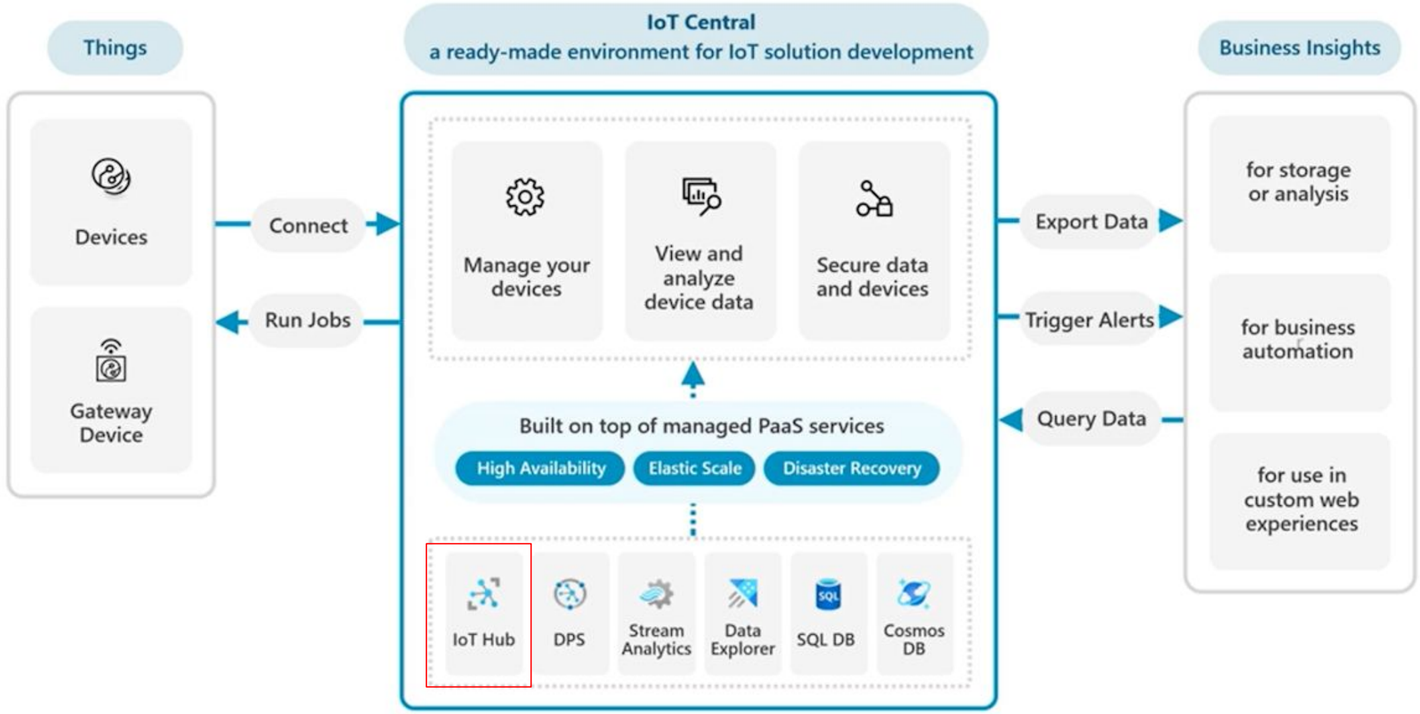


Una pluralità di servizi che “ci sono ma non si vedono”

https://learn.microsoft.com/en-us/azure/iot-central/core/concepts-architecture

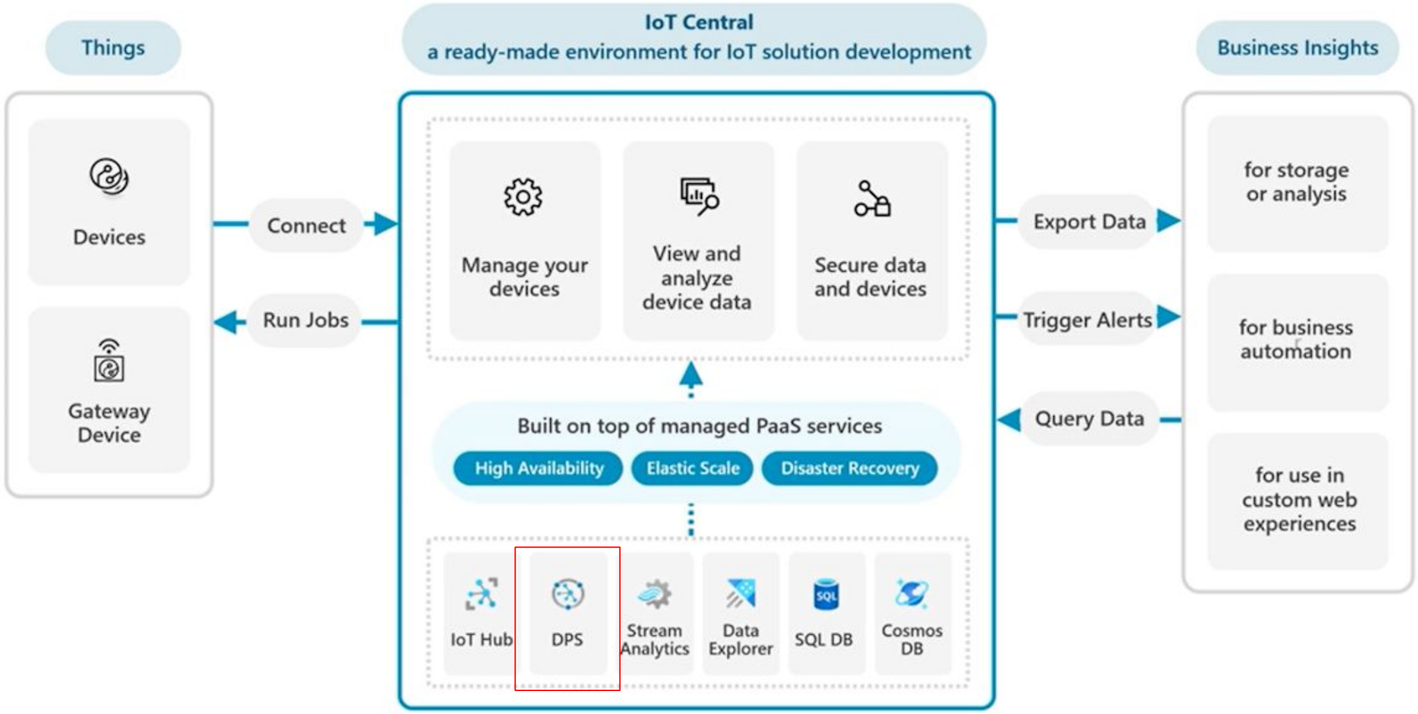
IoTCentral - Under the hood

- **IOTHub**: il sistema di comunicazione device-cloud e cloud-device;
  - permette di ricevere telemetrie, impostare proprietà, inviare comandi ai dispositivi.
  - IOTCentral ne usa più di uno, per motivi di scalabilità.



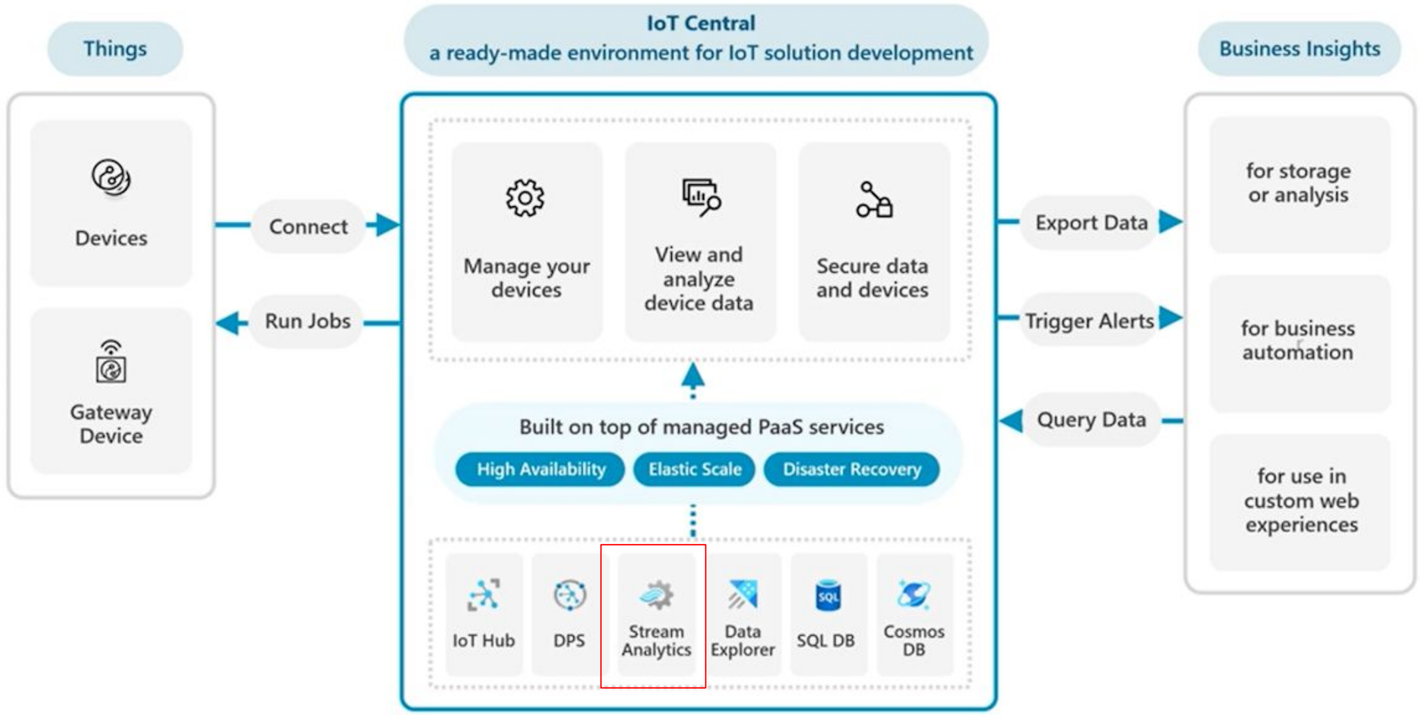
IoTCentral - Under the hood

- **DPS**: il servizio di provisioning dei device su iot-hub.
  - Assegna i device agli hub.
  - Verifica l'identità del device
  - Invia tutte le informazioni che servono per la connessione ad iot-hub



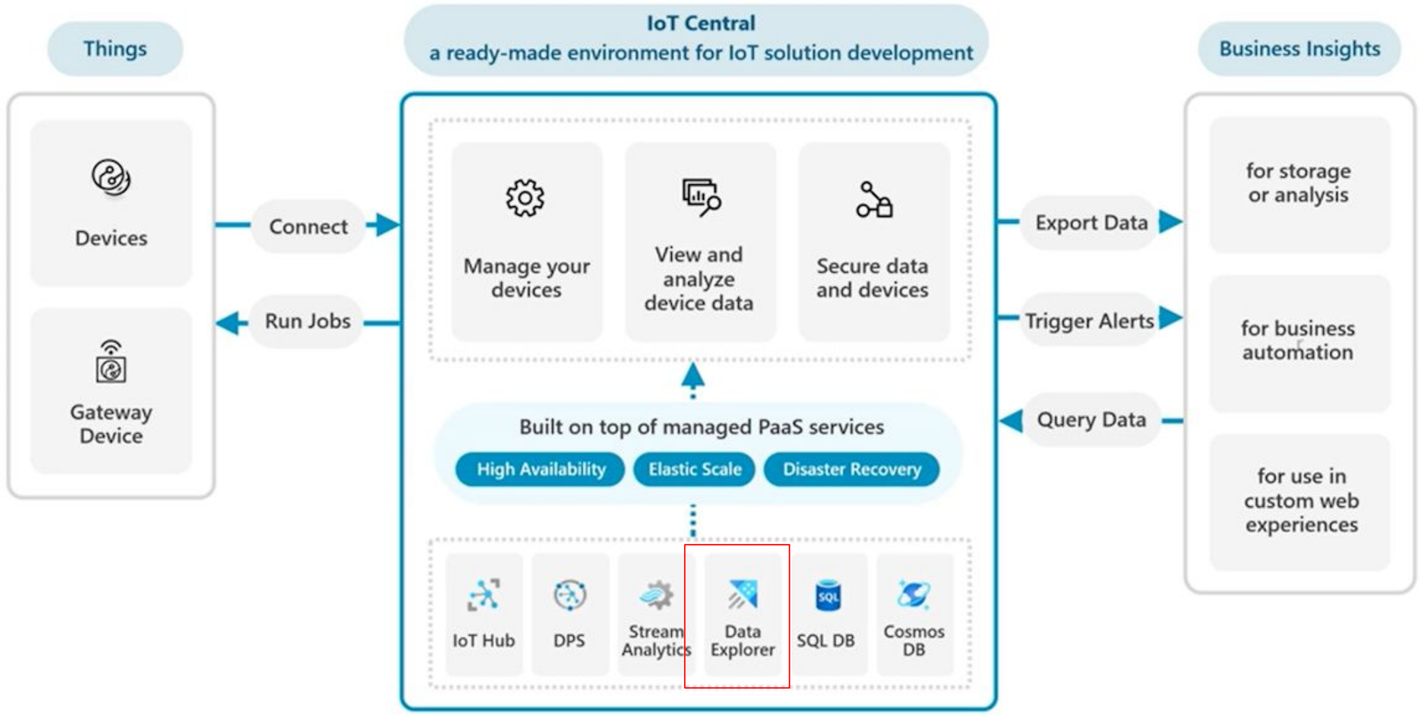
IoTCentral - Under the hood

- **Stream Analytics**: il servizio di analytics realtime per i dati (identificazione di pattern)



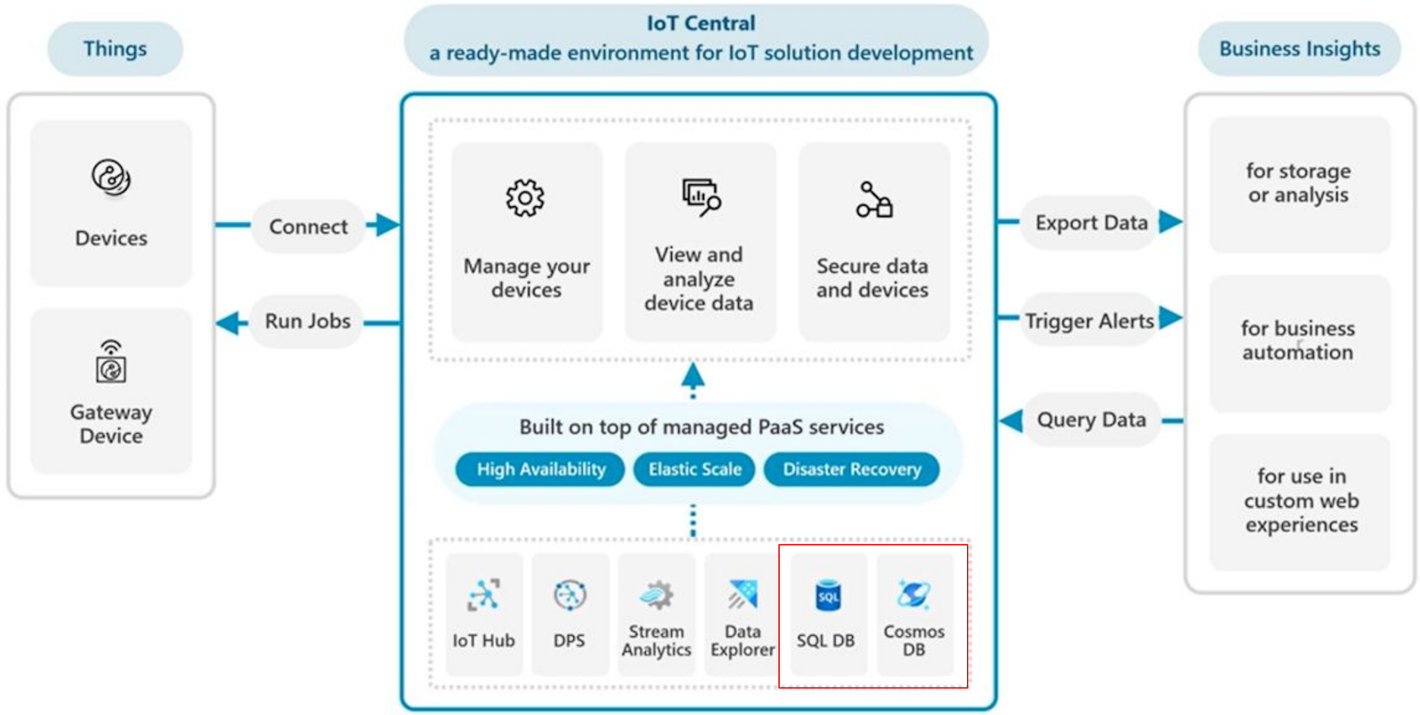
IoTCentral - Under the hood

- **Data Explorer**: è un servizio di analisi dei dati managed per l'esplorazione di dati (query).
  - sorgenti: applicazioni, siti Web, dispositivi IoT...



IoTCentral - Under the hood

- **SQLDB/CosmosDB**: i database per i dati.



Creare una applicazione IoTCentral

Dal portale Azure

Accessibile attraverso un link apposito come applicazione web a sé stante

IoT Central Application U... : https://iot-central-

Concetti fondamentali di IoTCentral

Device Template

Un blueprint che definisce caratteristiche e comportamenti del dispositivo.

Definisce un *Device Model* che specifica le tipologie di dati che vengono scambiati:

Per es:

- i valori che il dispositivo riceve
- i comandi che può eseguire

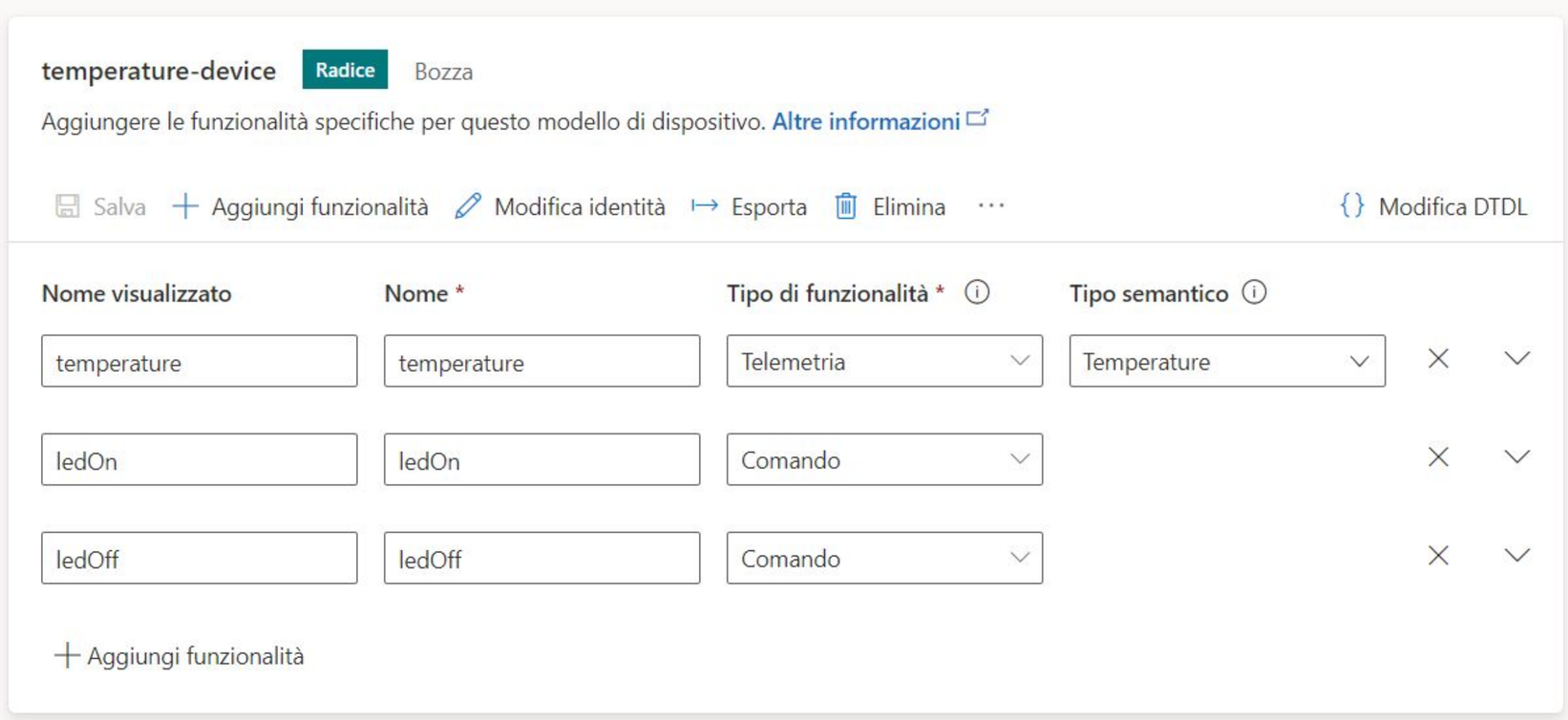
I dispositivi sono associati ad un template:

- tutti i dispositivi che “afferiscono” allo stesso template hanno lo stesso set di proprietà/telemetrie/comandi

Definizione del device template/device model

Avviene sul portale o via REST API

Possono anche essere autogenerati in base ad i dati che il dispositivo invia!



DTDL - Digital Twin Definition Language

Un linguaggio che definisce telemetrie, comandi, proprietà dei device.

- è una variante di JSON-LD: una estensione di JSON “per le ontologie”
  - definire degli “oggetti”
  - definire le loro relazioni
- IoTCentral usa una versione “estesa” di DTDL v2

Device

Crea un nuovo dispositivo

Per creare un nuovo dispositivo, selezionare un modello di dispositivo, un nome e un ID univoco.
Altre informazioni

Nome del dispositivo *

simulated_temperature_device

ID dispositivo *

Organizzazione *

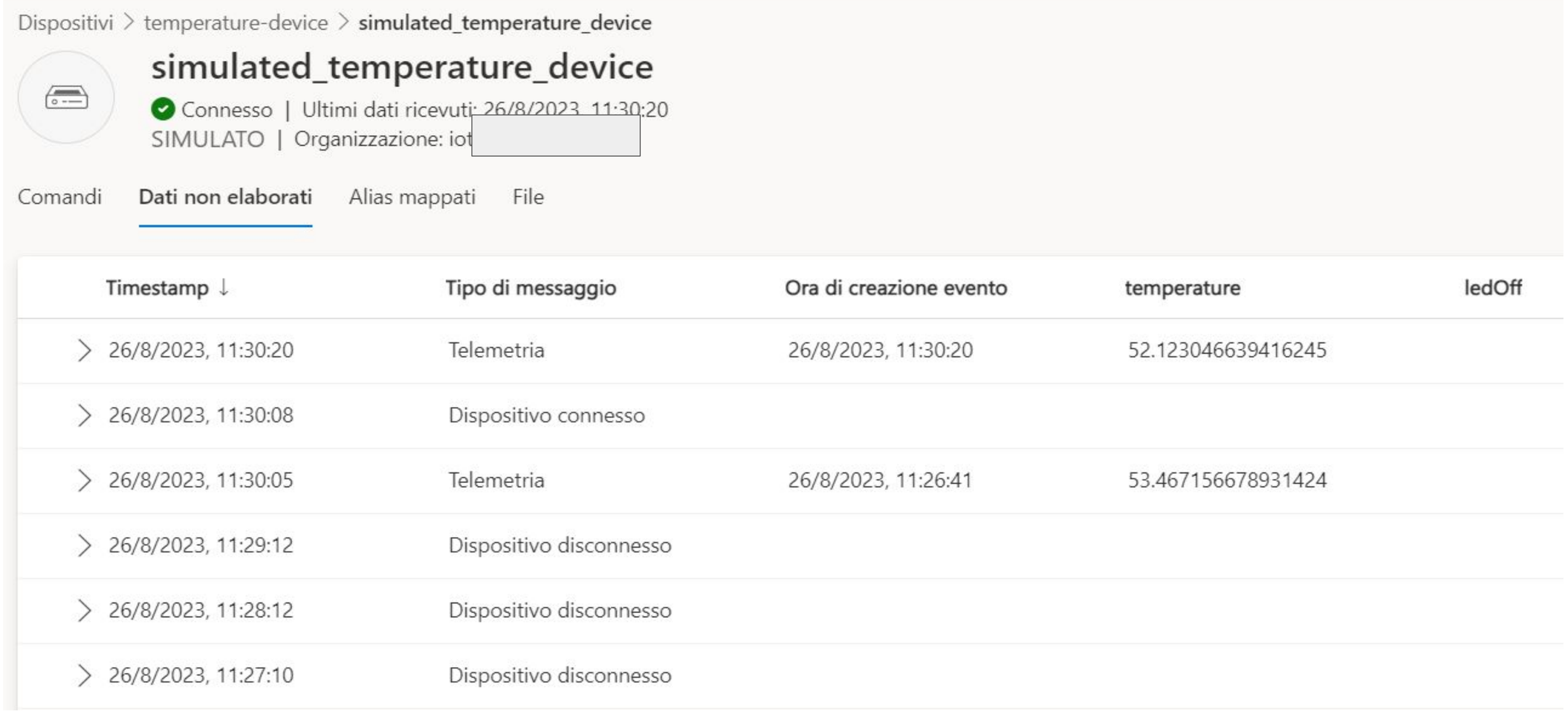
Modello di dispositivo *

temperature-device

Simulare questo dispositivo?
Un dispositivo simulato genera la telemetria che consente di testare il comportamento dell'applicazione prima di connettere un dispositivo reale.

Sì

Device simulati



Proprietà

Proprietà del device (non sono telemetrie!): valori di cui il device deve “avere contezza”

- read-only
  - es: la versione del firmware di un dispositivo

- writable
  - il device li riceve e poi segnala quando sono stati aggiornati (su se stesso)
  - es: la temperatura a cui deve rimanere un sensore

Telemetrie

Sono i valori che il dispositivo invia ad IoTCentral e che sono “monitorabili”.

Es:

- la temperatura rilevata
- l’umidità rilevata
- ...<qualsiasi altro valore da osservare>

Command

Il comando che il dispositivo può eseguire:

- possono essere “impartiti” da IoTCentral
  - via interfaccia
  - via REST API

Cloud Property

Sono proprietà che rimangono in IoTCentral e che non vengono mai sincronizzate con il dispositivo

Es:

- la data di ultimo intervento sul device

Inviare comandi dal portale Azure

Device simulato

Log dei comandi

Timestamp ↓	Tipo di messaggio	Ora di creazione evento
2/10/2023, 19:57:21	Risposta del comando	2/10/2023, 19:57:18
2/10/2023, 19:57:21	Richiesta comando	2/10/2023, 19:57:16

Log dei comandi - Dettagli

2/10/2023, 19:57:21 Risposta del comando 2/10/2023, 19:57:18


```
{
    "_eventcreationtime": "2023-10-02T17:57:18.442Z",
    "ledOff": null,
    "_eventtype": "Risposta del comando",
    "_timestamp": "2023-10-02T17:57:21.185Z"
}
```


2/10/2023, 19:57:21 Richiesta comando 2/10/2023, 19:57:16


```
{
    "_eventcreationtime": "2023-10-02T17:57:16.717Z",
    "ledOff": {
        "connectTimeoutInSeconds": 30,
        "methodName": "ledOff",
        "responseTimeoutInSeconds": 30
    },
    "_eventtype": "Richiesta comando",
    "_timestamp": "2023-10-02T17:57:21.122Z"
```

Dal device simulato al vero device

Registrazione del nuovo device (vero)

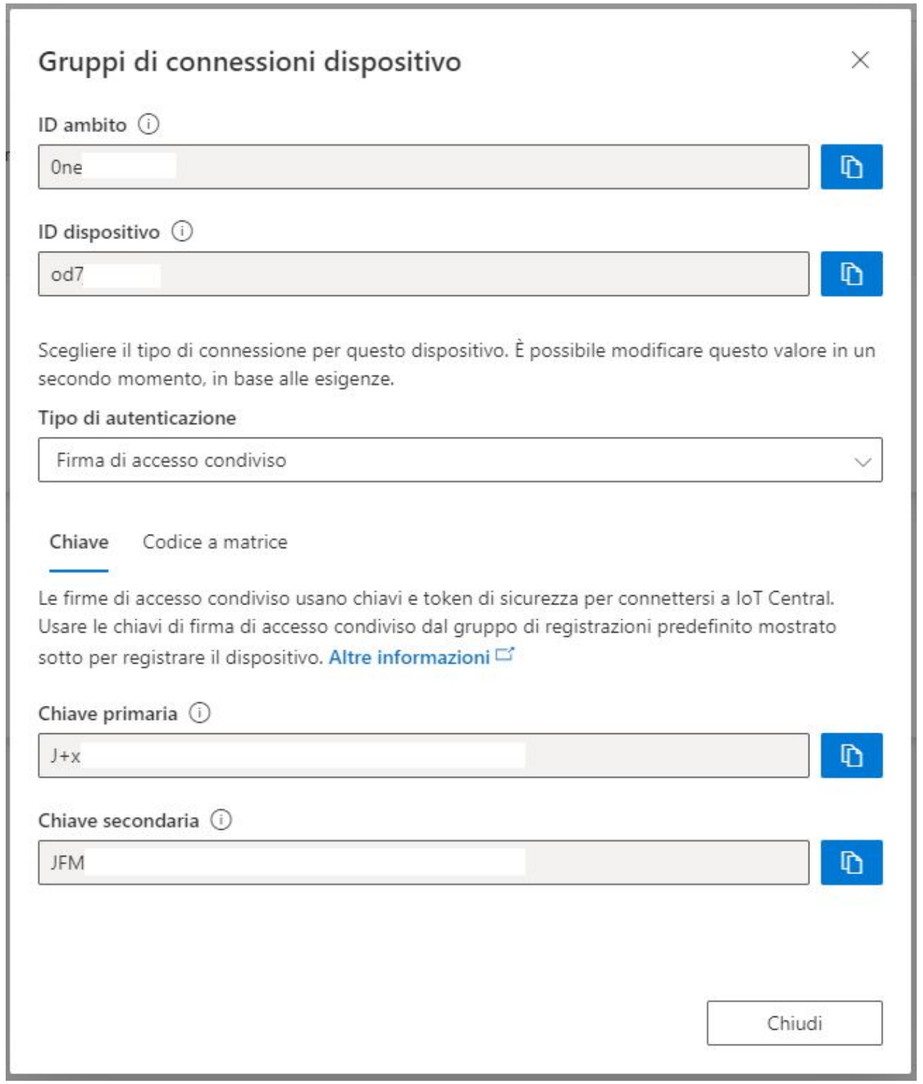
Permette di confermare l’identità del device

Sul portale:

- otteniamo le informazioni per la connessione (“Connetti”)

Altri tipi di registrazione:

- certificato
- TPM



Dal device simulato al vero device

Arduino Nano RP2040 Connect (MicroPython)

IDE:

- Thonny

Installazione: si carica lo script sul device

SDK MicroPython per IoTCentral

Sul device, facilita la connessione ad IoTCentral:

- per inviare telemetrie
- per ricevere comandi

https://github.com/Azure/iot-central-micropython-client

Sfide

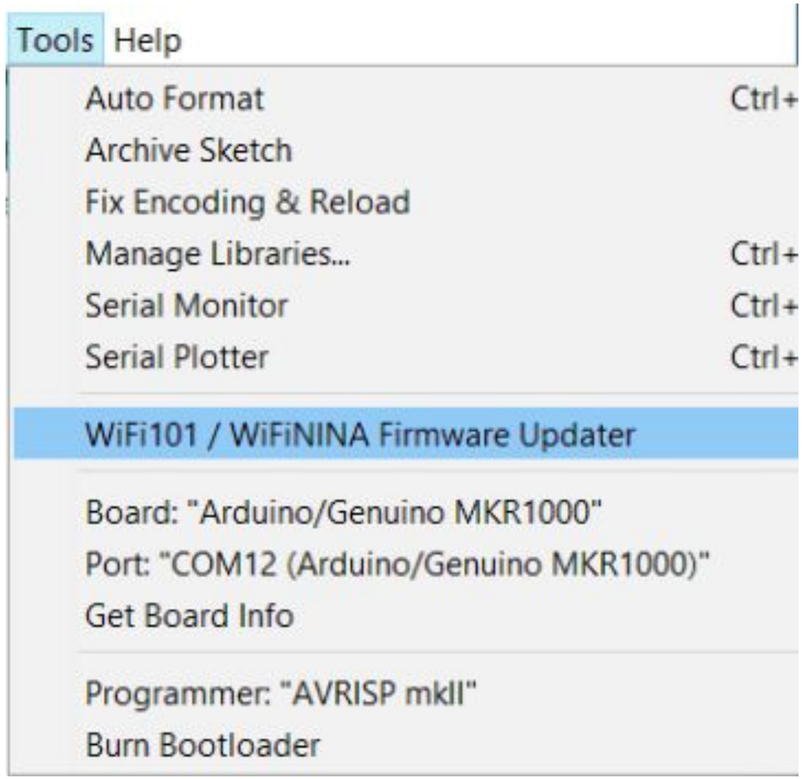
1. La versione di MicroPython che ho scaricato non arriva con ussl
2. Aggiornamento WiFiNINA

WiFiNINA: la libreria che permette la comunicazione di rete alla board

https://micropython.org/download/ARDUINO_NANO_RP2040_CONNECT/

Aggiornamento WiFiNINA

Da Arduino IDE, va fatto prima di caricare MicroPython



Ricompilare MicroPython

uSSL va integrato nel firmware

1. Clonare il repository di Micropyton

```
"set(MICROPY_PY_USSL 1)" in the mpconfigboard.cmake
```

```
"#define MICROPY_PY_USSL (1)" in the mpconfigboard.h
```

2. In WSL:

```
make -j$(nproc) -C src/micropython/mpy-cross
```

```
make -j$(nproc) TARGET=ARDUINO_NANO_RP2040_CONNECT -C src
```

https://stackoverflow.com/questions/74977591/how-can-i-get-the-ussl-module-for-my-micropython-project

Ricompilare MicroPython

Si perde la possibilità di interagire con OpenMV IDE

OpenMV IDE ha un suo firmware:

- nel quale si può includere ussl
- ...che poi è troppo ingombrante per la board :(

Thonny

Si può usare Thonny per lanciare lo script sulla board (per un test!)

Codice sul device

Package managing

```
try:
    import iotc
except:
    import mip
    mip.install('github:Azure/iot-central-micropython-client/package.json')
    import iotc
```

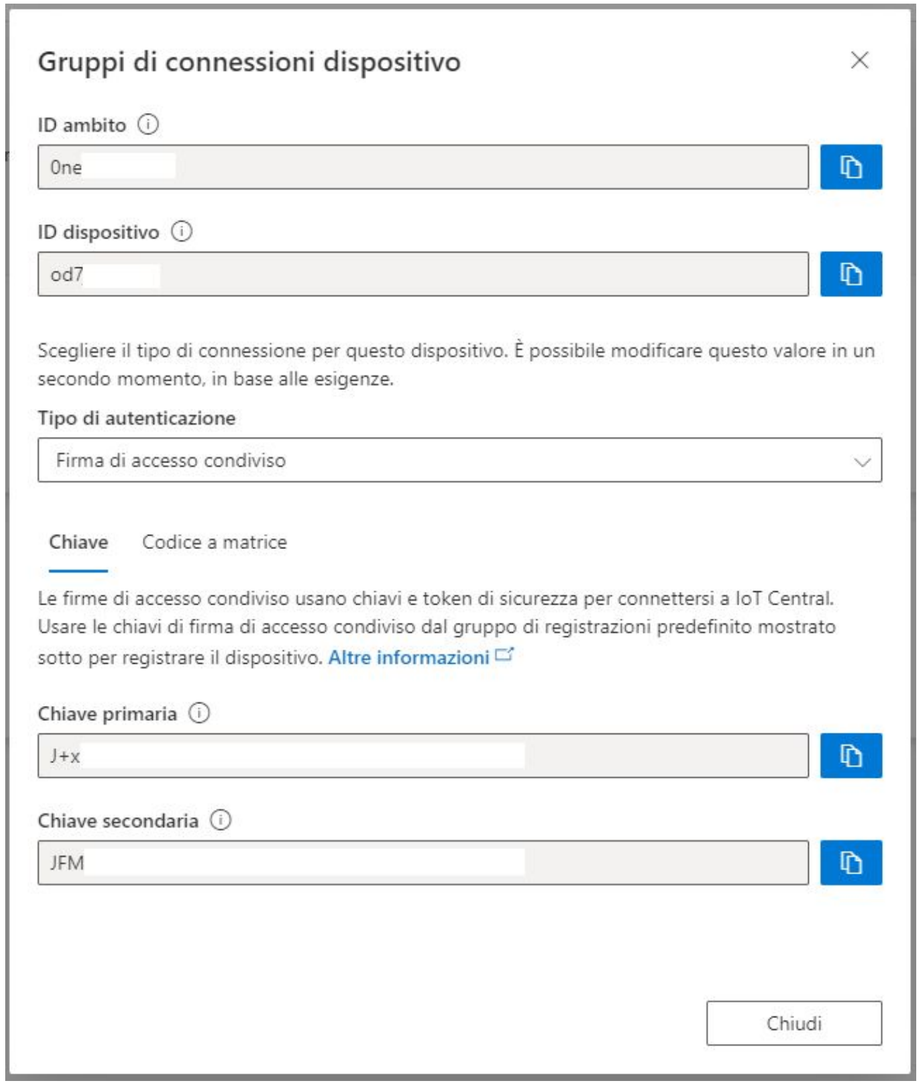
Codice sul device

```
import sys

import time

from random import randint

import network

from machine import Pin, I2C


temp_pin = 4

sensor = machine.ADC(temp_pin)

# led init

led = Pin(6, Pin.OUT)
```

Autenticazione

```
conn_type=IoTCConnectType.DEVICE_KEY
```

```
client=IoTCClient(scope_id,device_id,conn_type,key)

client.set_log_level(IoTCLogLevel.ALL)
```



Lettura della temperatura

```
def read_temperature():

    adc_value = sensor.read_u16()

    volt = (3.3/65535) * adc_value

    temperature = 27 - (volt - 0.706)/0.001721

    return round(temperature, 1)
```

Connessione

```
client.on(IoTCEvents.PROPERTIES, on_properties)

client.on(IoTCEvents.COMMANDS, on_commands)

client.connect()
```

Loop di lettura ed invio

```
start = time.ticks_ms()

while client.is_connected():

    client.listen()

    if time.ticks_diff(time.ticks_ms(), start) > 5000:

        print('Sending telemetry')

        temperature = read_temperature()

        client.send_telemetry({'temperature':temperature})

        start = time.ticks_ms()
```

Ricezione dei comandi

```
def on_commands(command, ack):
    ack(command, command.payload)
    if command.name == 'ledOn':
        led.on()
    if command.name == 'ledOff':
        led.off()
```

Loop di lettura ed invio (uasyncio)

```
while client.is_connected():
    client.listen()
    sleep(3)


loop = asyncio.get_event_loop()
loop.create_task(schedule_temp_read())
loop.run_forever()
```

Loop di lettura ed invio (uasyncio)

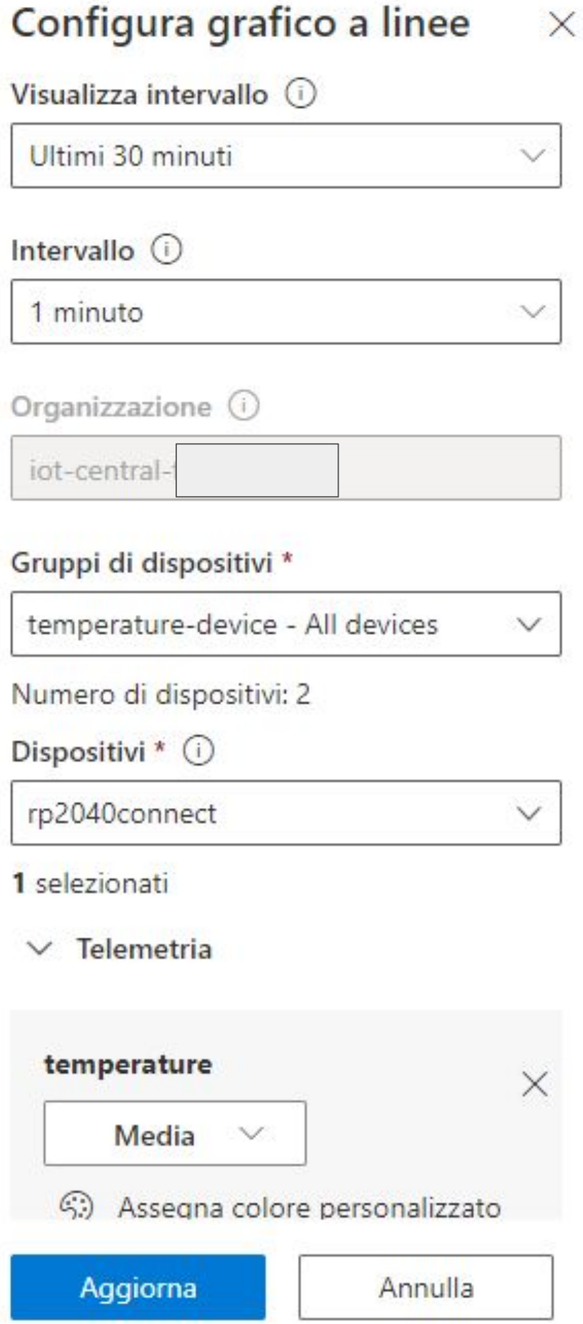
```
async def schedule_temp_read():
    while True:
        await asyncio.sleep(3600)
        print('Reading temperature')
        temperature = read_temperature()
        client.send_telemetry({'temperature':temperature})
```

Dashboard di IoTCentral

“Pannelli di controllo” self-service:

Es:

- ultimo valore rilevato
- valori rilevati nell’intervallo di tempo



Regole

Fanno “scattare dei trigger” al verificarsi di una condizione:

- es. la temperatura rilevata è maggiore di 30 gradi

Trigger:

- invio di una email
- chiamata a webhook
- gruppi di azioni
- interazione PowerAutomate
- interazione PowerApps

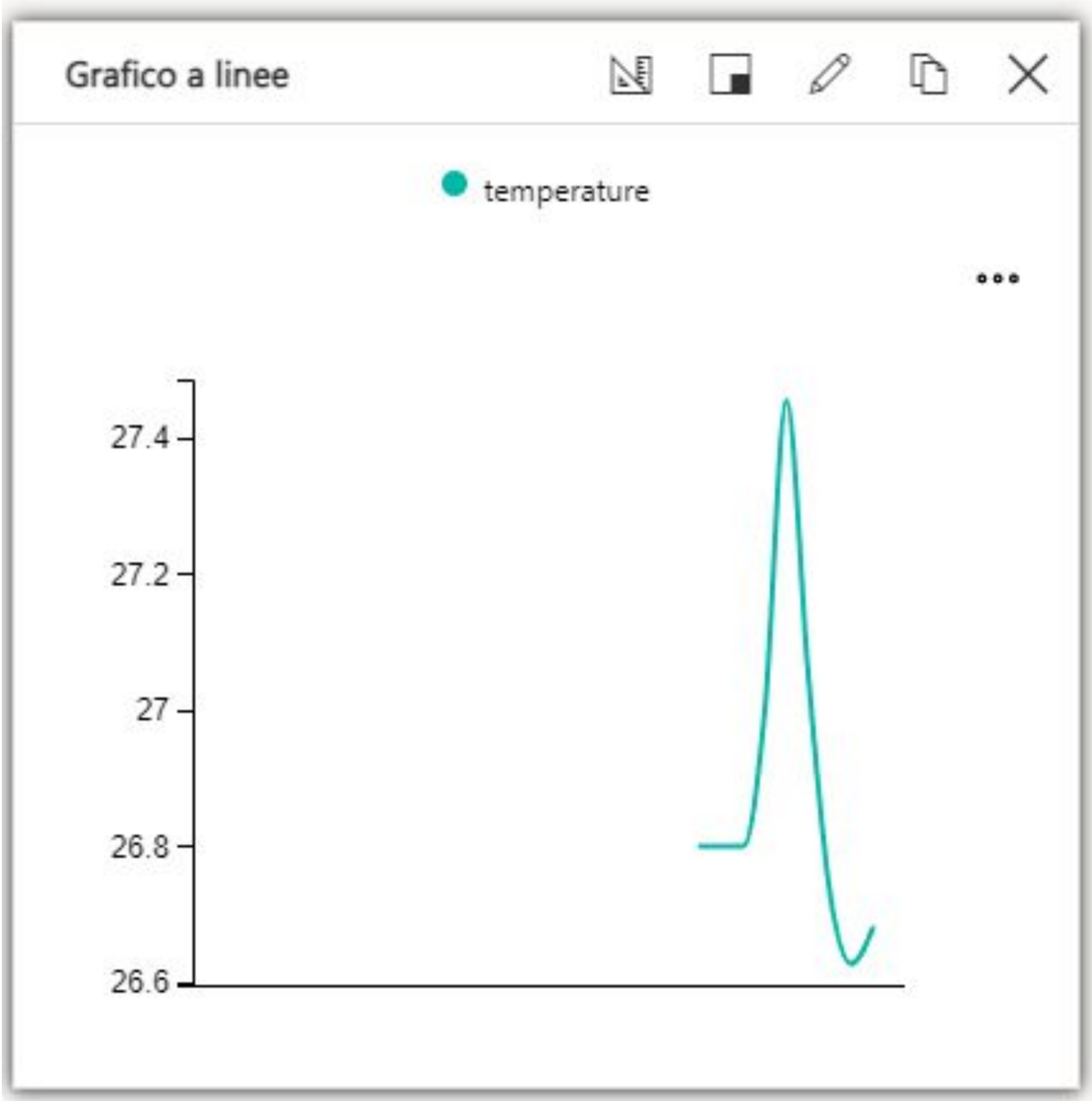
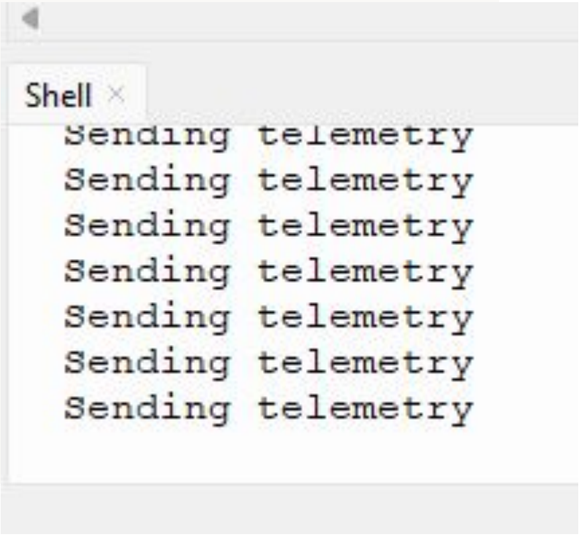


Grafico a linee
temperature
27.4
27.2
27
26.8
26.6


Ultimo valore noto
26,60
ora


Shell
Sending telemetry
Sending telemetry
Sending telemetry
Sending telemetry
Sending telemetry
Sending telemetry
Sending telemetry

Usare le REST API

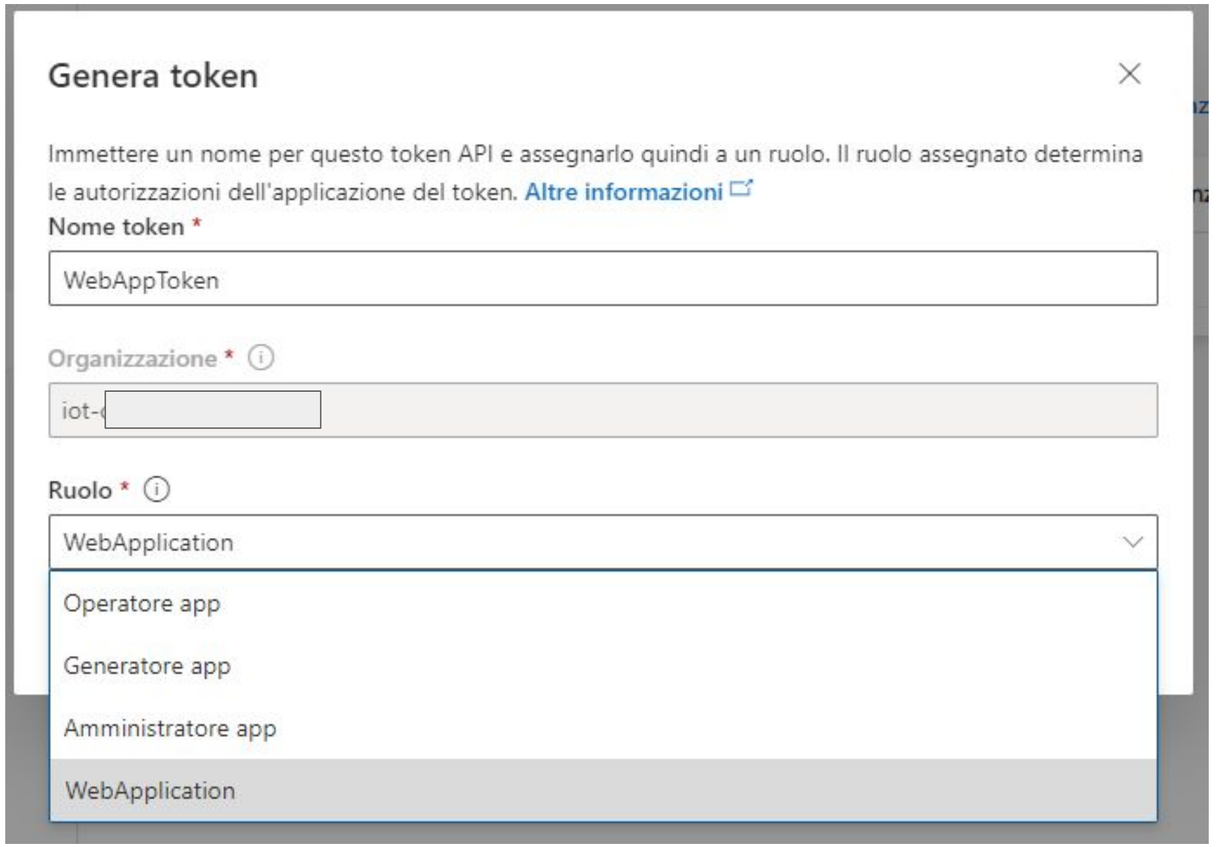
Usare le REST API

Avere una “propria” webapp per:

- leggere telemetrie
- inviare comandi

Autorizzazione

WebApplication	Applicazione	Accesso via applicazione web



dataclasses/dataclasses_json

dataclasses_json: un package che permette di serializzare/deserializzare dataclasses

L'idea:

- deserializzare oggetti di IoTCentral dalle response
  - device
  - command
  - device template
  - properties

Device

```python
def json_response(cl):
    return dataclass_json(dataclass(cl))

@json_response
class Device:
    id: str
    etag: str
    displayName: str
    simulated: bool
    provisioned: bool
    template: str
    enabled: bool
```

Device

Definizione del dispositivo.

Nome	Tipo	Descrizione
displayName	string	Nome visualizzato del dispositivo.
enabled	boolean	Indica se la connessione del dispositivo a IoT Central è stata abilitata.
etag	string	ETag usato per evitare conflitti negli aggiornamenti dei dispositivi.
id	string	ID univoco del dispositivo.
organizations	string[]	Elenco di ID organizzazione di cui fa parte il dispositivo, attualmente è supportata una sola organizzazione, più organizzazioni saranno presto supportate.
provisioned	boolean	Indica se le risorse sono state allocate per il dispositivo.
simulated	boolean	Indica se il dispositivo è simulato.
template	string	Definizione del modello di dispositivo per il dispositivo.

Command

```
@json_response
class Command:
    id: Optional[str] = field(metadata=config(field_name="@id"))
    type: str=field(metadata=config(field_name="@type"))
    name: str
    description: Optional[str] = None
    displayName: Optional[str] = None
    commandType: Optional[str] = None
    comment: Optional[str] = None
```

Device Template

```python
@json_response
class DeviceTemplate:
    etag: str
    displayName: str
    capabilityModel: CapabilityModel
    id: str = field(metadata=config(field_name="@id"))
    type: list[str] = field(metadata=config(field_name="@type"))
    context: list[str] = field(metadata=config(field_name="@context"))
```

Capability Model

```python
@json_response
class CapabilityModel:
    type: str = field(metadata=config(field_name="@type"))
    id: str = field(metadata=config(field_name="@id"))
    contents: list[Command | Telemetry | Property | CloudProperty]
    displayName: str
    description: Optional[str] = None
```

Service

“Lavora” il prodotto delle chiamate delle API REST per fornire degli oggetti “completi”.

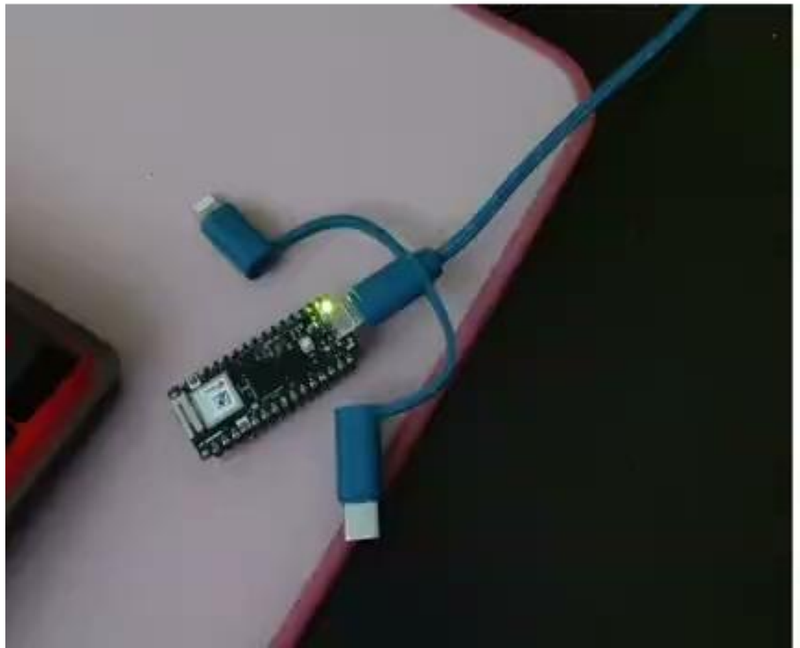
```
complete_devices = list()
devices = self.IOTCentralAPI.get_devices()
for device in devices.value:
    device_template = self.IOTCentralAPI.get_template(device.template)
    device_capabilities = self.device_objects_by_type(device_template.capabilityModel.contents)
    complete_device = CompleteDevice(
        name=device.id,
        display_name=device.displayName,
        commands=device_capabilities['Command'],
        telemetries=device_capabilities['Telemetry'],
        cloud_properties=device_capabilities['CloudProperties'],
        properties=device_capabilities['Properties'])
    complete_devices.append(complete_device)
return complete_devices
```

Utilizzo del wrapper

```
APP_SUBDOMAIN = os.getenv('APP_SUBDOMAIN')
SAS_TOKEN = os.getenv('SAS_TOKEN')

iotc = IOTCentral(
    app_subdomain=APP_SUBDOMAIN,
    auth_type=AuthType.SAS_TOKEN,
    token=SAS_TOKEN
)

# listing devices and properties/commands/telemetries/cloud properties
devices: list[CompleteDevice] | IOTCentralError = iotc.get_devices()  # noqa: F821
for device in devices:
    print(device.commands)
    print(device.telemetries)

res = iotc.send_command(<device_id>,'ledOff')
```

Devices

- rp2040connect led_on led_off



Grazie!