UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA

Corso di Laurea in
INGEGNERIA INFORMATICA E DELL'AUTOMAZIONE

Dipartimento di Ingegneria dell'Informazione

ELABORAZIONE DEI DATI DI UN SENSORE IOT SU PIATTAFORMA CLOUD

DATA PROCESSING OF AN IOT SENSOR ON A CLOUD PLATFORM

Relatore: Prof.
ENNIO GAMBI

Tesi di Laurea di:
ROBERTO PIANELLA

A.A. 2019/2020

INDICE

CAPITOLO 1: INTERNET OF THINGS

CAPITOLO 2: PROTOCOLLI DI COMUNICAZIONE

CAPITOLO 3: PIATTAFORME IOT

CAPITOLO 1:
INTERNET OF THINGS

Il termine "*Internet of Things*" (**IoT**) fu usato per la prima volta, nel 199, dall'imprenditore tecnologico britannico Kevin Ashton per descrivere un sistema nel quale gli oggetti situati nel mondo fisico possono essere collegati a Internet attraverso sensori. Ashton coniò il termine mentre lavorava con dispositivi RFID[1] per illustrare la potenza di collegare "*tags*"[2] RFID a internet in modo da tracciare i beni senza l'intervento umano.

Oggi l'*Internet of Things* è diventato un termine popolare che descrive scenari in cui la connessione Internet e la capacità computazionale sono estese a molti oggetti.

In questo capitolo verrà introdotto l'IoT, i vari modi con cui può essere definito e i modelli di comunicazione.

1.1. Cos'è l'Internet of Things (IoT)

IoT, semplificando, è tutto ciò che è connesso ad una rete (compreso Internet) o ad altre macchine, lavorando autonomamente senza necessità di intervento umano. Tutti gli altri termini descrivono concetti che sono rese possibili dall'*Internet of Things*.

Nel privato termini come ad esempio: *Connected Home/Car*, etc., significano semplicemente che sono in qualche modo oggetti connessi in rete.

Lo stesso vale per l'industria. Questa connettività, resa possibile da un insieme di protocolli wireless e dalla moderna componentistica, permette ai progettisti di realizzare macchinari ed equipaggiamenti "*smart*" che permettono di tracciarsi, monitorarne il funzionamento, visualizzare dati e regolarsi di conseguenza in modo autonomo.

Se l'idea di connettere gli oggetti tra di loro non è nuova, bisogna chiedersi, "Perchè l'*Internet of Things* è così popolare oggi?"

Da un'ampia prospettiva possiamo dire che la confluenza di diverse tecnologie e tendenze di mercato ha reso possibile connettere, in modo accessibile, sempre più dispositivi a basso costo.

[1] **Radio-frequency identification (RFID)**: Sistema che utilizza campi elettromagnetici per identificare e tracciare automaticamente i tag attaccati agli oggetti. RFID è un metodo di identificazione automatica e acquisizione dati (AIDC).

[2] **Tags**: Tipo di sistema di tracciamento che utilizza codici a barre intelligenti per identificare gli articoli. I tag contengono informazioni archiviate elettronicamente.
I **tag passivi** raccolgono energia dalle onde radio interroganti di un lettore RFID nelle vicinanze.
I **tag attivi** hanno una fonte di alimentazione locale (come una batteria) e possono funzionare a centinaia di metri dal lettore RFID.
A differenza di un codice a barre, i tag non devono trovarsi all'interno della linea di vista del lettore, quindi potrebbe essere incorporato nell'oggetto tracciato.

Possiamo racchiudere questa crescita esponenziale in sei punti chiave:

- **Ubiquità della connessione**: basso costo, alta velocità, pervasività della connessione di rete, rende quasi tutto "connettibile".
- **Diffusa adozione di reti *"IP-based"***: L'IP[3] è diventato lo standard globale dominante per la rete.
  Fornisce una piattaforma ben definita e ampiamente implementata di software e strumenti che possono essere incorporati in una vasta gamma di dispositivi facili da utilizzare ed economici.
- **Capacità di calcolo economica**: guidata dai grandi investimenti delle industrie nella ricerca, sviluppo e produzione, la legge di Moore[4] continua a fornire una maggior potenza di calcolo a basso costo e basso consumo energetico.
- **Miniaturizzazione:** i progressi costruttivi permettono di incorporare la capacità di calcolo all'avanguardia e la comunicazione in dispositivi molto piccoli. Questo, accoppiato con la capacità di calcolo a basso costo, ha alimentato l'avanzamento di sensori piccoli ed economici che guidano molte applicazioni IoT.
- **Avanzamento nell'analisi dei dati**: I nuovi algoritmi e la rapida crescita della potenza di calcolo, spazio di archiviazione e i servizi cloud hanno abilitato l'aggregazione, la correlazione e l'analisi di grandi quantità di dati. Questi insiemi di informazioni, vasti e dinamici, forniscono nuove opportunità di estrarre conoscenza.
- **Ascesa della computazione cloud**: La computazione cloud sfrutta l'elaborazione remota di processi in rete (gestione e archiviazione di dati) permettendo a piccoli dispositivi di interagire attraverso un potente "*back-end*" con ottime capacità analitiche.

In un report del McKinsey Global Institute[5] intitolato "*Unlocking the Potential of the Internet of Things",* viene descritta l'ampia gamma di potenziali applicazioni in termini di "settore" in cui si prevede che l'IoT crei un valore per l'industria e per gli utenti.
Di seguito viene mostrato il riassunto del suddetto report.

[3]**IP**: Etichetta numerica assegnata a ciascun dispositivo collegato a una rete di computer che utilizza il protocollo Internet per la comunicazione. Un indirizzo IP svolge due funzioni principali: identificazione dell'interfaccia host o di rete e indirizzamento della posizione.
Internet Protocol versione 4 (IPv4) definisce un indirizzo IP come un numero a 32 bit. Tuttavia, a causa della crescita di Internet e dell'esaurimento degli indirizzi IPv4 disponibili, nel 1998 è stata standardizzata una nuova versione di IP (IPv6), che utilizza 128 bit per l'indirizzo IP.

[4]**Legge di Moore**: Legge empirica che descrive lo sviluppo della microelettronica, con una progressione sostanzialmente esponenziale. Essa afferma che la complessità dei microcircuiti raddoppia periodicamente, con un periodo di 18 mesi.

[5]**McKinsley Global Institute**: Complesso istituito nel 1990 per sviluppare una comprensione più profonda dell'evoluzione dell'economia globale.

Settore	Descrizione	Esempio
Umano	Dispositivi all'interno e/o all'esterno del corpo umano	Dispositivi (indossabili e/o ingeribili) per monitorare salute, benessere, malattie, fitness ecc.
Casa	Edifici Residenziali	*Home Automation* e sistemi di sicurezza.
Uffici	Spazi di lavoro	Gestione dell'energia e sicurezza negli uffici, miglioramento della produttività.
Veicoli	Sistemi all'interno di veicoli	Veicoli che comprendono automobili, camion, navi, aerei, treni, includendo un sistema di manutenzione, statistiche, posizione, ecc.
Città	Ambienti urbani	Spazi pubblici e infrastrutture urbane, adozione di controllo del traffico, contatori intelligenti, gestione delle risorse.
Negozi	Spazi dove i consumatori commerciano	Negozi, banche, ristoranti, stadi, ovunque il consumatore compri qualcosa, possibilità di effettuare il pagamento senza un commesso, avviso di sconti all'interno dei negozi, ottimizzazione dell'inventario.

1.2. Architettura IoT

L'architettura dell'IoT è basata su tre concetti:

Things: I dispositivi, "*cose*", connessi via cavo o wireless ad una rete

Network: Il network connette tutte le "*cose*" di una rete al cloud

Cloud: I server di un "*data center*" remoto che immagazzinano i tuoi dati in modo sicuro

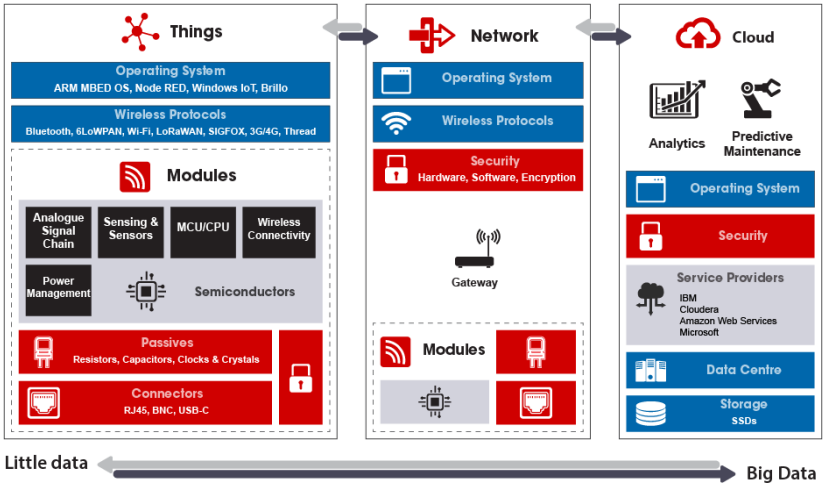


Figura 1.1: Schematizzazione dell'IoT

1.3. Data

Le *"cose"* generano dati/gruppi di dati racchiusi in pochi byte che rappresentano informazioni provenienti da sensori che possono essere, ad esempio : sensori di temperatura, di umidità e di posizione.

Mentre i dispositivi inviano piccoli pacchetti di dati nel cloud attraverso il network questi vengono consolidati e tracciati, diventando sempre più grandi nel tempo.
Questo viene descritto come *"big data"* ed è qui che l'IoT diventa veramente importante. Le informazioni contenute nei *"big data"* permettono all'utente di esaminare migliaia/milioni di dati per capire/controllare una determinata variabile, un processo o un sistema, nel modo migliore.

Ad esempio: sapendo che in primavera fa buio più tardi, utilizzando i dati provenienti da fotocellule piazzate sui lampioni stradali è possibile fare in modo di ritardarne l'accensione risparmiando, di conseguenza, energia.
Un altro esempio può essere: Tramite dei sensori su strada è possibile individuare un'anomalia nel funzionamento dell'autovettura permettendo, quindi, di pianificare interventi di manutenzione preventiva evitando dunque guasti improvvisi.

Dunque i dati immagazzinati possono essere utilizzati sia come oggetto di semplice studio sia come "*triggers*"[6] di altre azioni conseguenti.

1.4. Modelli di comunicazione

Nel Marzo 2015, l'*Internet Architecture Board*[7] ha rilasciato il documento **RFC 7452**[8], che spiega l'architettura di rete degli *"smart object"*, delineando un quadro di quattro modelli di comunicazione utilizzati dai dispositivi IoT.

1.4.1. Comunicazione Device-to-Device

Il modello di comunicazione "*Device-to-Device*" (**D2D**) rappresenta due o più dispositivi che sono connessi direttamente e comunicano tra di loro, evitando di utilizzare un server come intermediario.
Questi dispositivi scambiano informazioni attraverso diversi tipi rete che includono reti IP e Internet. Spesso questo tipo di comunicazione utilizza protocolli come Bluetooth[9], Z-Wave[10] o ZigBee[11] per stabilire una connessione "*Device-to-Device*" come mostrato in figura.

[6]**Trigger**: Nelle basi di dati, procedura che viene eseguita automaticamente in coincidenza di un determinato evento.

[7]**Internet Architecture Board (IAB)**: Organizzazione pubblicitaria che sviluppa standard di settore, conduce ricerche e fornisce assistenza legale per il settore della pubblicità online.

[8]**RFC 7452**: Codice identificativo del documento "*Architectural Considerations in Smart Object Networking*"

[9]**Bluetooth:** Standard tecnico-industriale di trasmissione dati per reti personali senza fili.

[10]**Z-Wave**: Rete mesh che utilizza onde radio a bassa energia per comunicare da un apparecchio.

[11]**ZigBee**: Rete mesh wireless a basso costo, a bassa potenza, destinato a dispositivi alimentati a batteria in applicazioni di controllo e monitoraggio wireless.

Figura 1.2: Esempio di comunicazione "*Device-to Device*".

Il modello "*Device-to-Device*" è comunemente usato in applicazioni come sistemi di *home automation*, che tipicamente utilizzano piccoli pacchetti di dati per la comunicazione tra i dispositivi. Per esempio dispositivi IoT come lampadine, interruttori (di luce), termostati, serrature, ecc. scambiano piccole informazioni tra loro per effettuare operazioni in uno scenario di *home automation* (Es.: Il messaggio di accensione o di spegnimento che può ricevere una lampadina è veramente piccolo).

Questo tipo di comunicazione generalmente, avendo una diretta relazione tra i dispositivi, ha bisogno di impostazioni di sicurezza, al loro interno, in modo da creare un meccanismo di fiducia tra i dispositivi. Questo significa che i produttori di questi dispositivi hanno bisogno di effettuare maggiori investimenti in termini di sicurezza per realizzare questi approcci e per implementare uno specifico formato di dati rispetto ad architetture più aperte.

Dal punto di vista dell'utente, invece, sorgono problemi di compatibilità forzando l'utente a scegliere una famiglia di dispositivi che utilizza uno specifico protocollo. (Es.: i dispositivi che utilizzano Z-Wave non sono nativamente compatibili con quelli della famiglia ZigBee)

1.4.2. Comunicazione Device-to-Cloud

Altro modello è il "*Device-to-Cloud*" (**D2C**), nel quale il dispositivo IoT si connette direttamente a un servizio Internet cloud.

Questo approccio utilizza metodi di comunicazione tradizionali come connessioni Wi-Fi[12] o Ethernet[13] per stabilire una connessione tra il dispositivo e la rete IP connessa al cloud.

[12]**Wi-Fi**: Tecnologia per reti locali senza fili (WLAN) che utilizza dispositivi basati sugli standard IEEE 802.11.

[13]**Ethernet**: Famiglia di tecnologie standardizzate per reti locali, sviluppata a livello sperimentale da Robert Metcalfe e David Boggs (suo assistente) allo Xerox PARC, che ne definisce le specifiche tecniche a livello fisico (connettori, cavi, tipo di trasmissione, etc.) e a livello MAC del modello architetturale di rete ISO/OSI.
Più in generale è utilizzata nelle reti locali (LAN), nelle reti metropolitane (MAN) e nelle reti geografiche (WAN). È stato commercializzata nel 1980 e inizialmente standardizzata nel 1983 come IEEE 802.3, e da allora ha mantenuto una buona dose di retrocompatibilità ed è stato perfezionato per supportare velocità in bit più elevate e distanze di collegamento più lunghe.

[14]**Termostato Nest**: Dispositivo che trasmette i dati a un database cloud dove vengono analizzati per verificare i consumi energetici. Inoltre la connessione con il cloud permette all'utente di utilizzare il termostato da remoto tramite smartphone o interfaccia web, oltre a consentire gli update software.

[15]**Gateway**: Dispositivo di rete che collega due reti informatiche di tipo diverso operando sia al livello di rete che ai livelli superiori, del modello ISO/OSI. Il suo scopo principale è quello di veicolare i pacchetti di rete all'esterno di una rete locale.

[16]**Timestamp:** Sequenza di caratteri che rappresentano una data e/o un orario per accertare l'effettivo avvenimento di un certo evento.

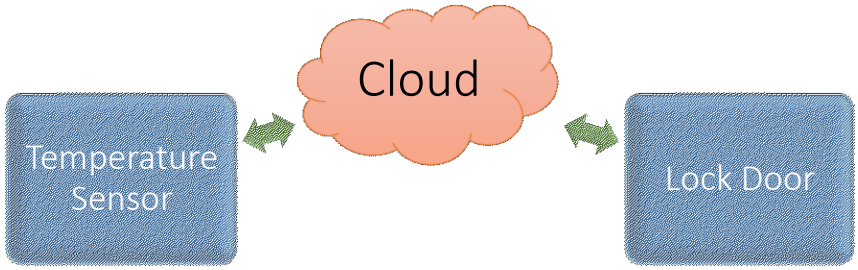


Figura 1.3: Esempio di comunicazione "*Device-to-Cloud*".

Il D2C è implementato da alcuni dispositivi IoT famosi sul mercato, come per esempio il Termostato Nest[14].

Tuttavia se si cerca di integrare dispositivi e cloud di differenti produttori ci si può trovare di fronte a problemi di incompatibilità portando quindi il consumatore ad utilizzare interfacce web o app differenti per ogni dispositivo di marca differente.

1.4.3. Comunicazione Device-to-Gateway

Nel modello di comunicazione "*Device-to-Gateway*" (**D2G**) il dispositivo IoT si connette attraverso un Gateway[15] al servizio cloud.

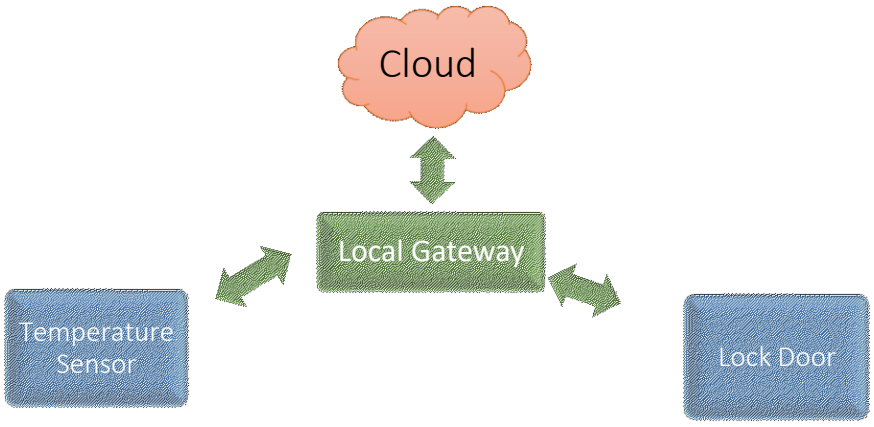


Figura 1.4: Esempio di comunicazione "*Device-to-Gateway*".

Il Gateway in questa configurazione assume tre ruoli principali.

Il primo è quello di trasformare i dati e normalizzarli.
Per esempio se il sensore non ha capacità di computazione e l'unica cosa che può fare è spedire il dato "grezzo" sarà il gateway che completerà l'informazione aggiungendo il "*timestamp*"[16] e il nome del dispositivo che l'ha

generato. Oppure i dati generati dai sensori possono essere in diversi formati e quindi è il gateway che si occupa di normalizzarli in un unico formato.
Il Gateway quindi acquisisce dati eterogenei provenienti dai vari dispositivi e li converte in un formato standard che sarà comprensibile per la fase successiva nella elaborazione dei dati.

Il secondo ruolo del Gateway è quello di supportare diversi protocolli di comunicazione, perché dispositivi IoT di diversi produttori molto probabilmente utilizzano protocolli di comunicazione differenti.
Quindi il Gateway deve supportare protocolli diversi per le connessioni in arrivo, ma anche per le connessioni in uscita, poiché esso si connette al Cloud.(tra i più popolari protocolli usati in questo contesto sono: ReST[17], MQTT[18], CoAP[19], STOMP[20] e SMS[21]).
In alcuni casi un Gateway può anche processare i dati ed emettere avvisi in tempo reale, quindi non far arrivare una specifica informazione al cloud se è stato così impostato per alcuni parametri.

Il terzo e ultimo ruolo, non per importanza, è quello di gestire la sicurezza.
Il gateway lavora come dispositivo di connessione proteggendo i dispositivi IoT dalla rete pubblica e incrementando notevolmente la sicurezza.

1.4.4. Comunicazione Back-End Data-Sharing

Il modello di condivisione di dati *"Back-End"* (**BSDM**) si riferisce ad un'architettura di comunicazione che consente agli utenti di esportare e analizzare i dati degli oggetti intelligenti da un servizio cloud in combinazione con i dati provenienti da altre fonti, permettendo così (a terzi) l'accesso ai dati dei sensori caricati.

Questo approccio è un'estensione del modello di comunicazione D2C, che può portare a silos di dati[22] in cui i dispositivi IoT caricano i le informazioni accumulate solo su un singolo fornitore di servizi applicativi.
Consente, quindi, di aggregare e analizzare i dati raccolti da singoli flussi del dispositivo IoT.

Ad esempio, se un utente aziendale incaricato di un complesso di uffici fosse interessato a consolidare e analizzare i dati sul consumo di energia e sui servizi di pubblica utilità, prodotti da tutti i sensori IoT e dai sistemi di utilità abilitati a Internet in loco, un'efficace architettura

[17]**Representational State Transfer (REST)**: Stile architetturale (di architettura software) per i sistemi distribuiti. L'architettura REST si basa su HTTP; il funzionamento prevede una struttura degli URL ben definita (atta a identificare univocamente una risorsa o un insieme di risorse).

[18]**MQTT**: vedi Capitolo 2 Paragrafo 1.

[19]**CoAP**: vedi Capitolo 2 Paragrafo 2.

[20]**Simple (o Streaming) Text Oriented Message Protocol (STOMP)**: Protocollo testuale, progettato per funzionare con middleware orientato ai messaggi (MOM). Fornisce un formato di filo interoperabile che consente ai client STOMP di parlare con qualsiasi broker di messaggi che supporta il protocollo.

[21]**Short Message Service (SMS)**: Componente del servizio di messaggistica di testo della maggior parte dei sistemi di telefonia, Internet e dispositivi mobili.

[22]**Silos di dati:** Archivio di dati fissi che rimane sotto il controllo di un'organizzazione ed è isolato dal resto della stessa.

di condivisione dei dati di back-end consentirebbe all'azienda di accedere e analizzare facilmente i dati nel cloud prodotti dall'intero spettro di dispositivi nell'edificio.

Inoltre, questo modello facilita le esigenze di portabilità dei dati, consente agli utenti di spostare le proprie informazioni quando passano da un servizio IoT all'altro, abbattendo le tradizionali barriere dei silos di dati.

Il BSDM suggerisce che per ottenere l'interoperabilità dei dati dei dispositivi *"smart"* ospitati nel cloud sono necessari un approccio di servizi cloud federati o interfacce per programmatori di applicazioni cloud (API)[23]

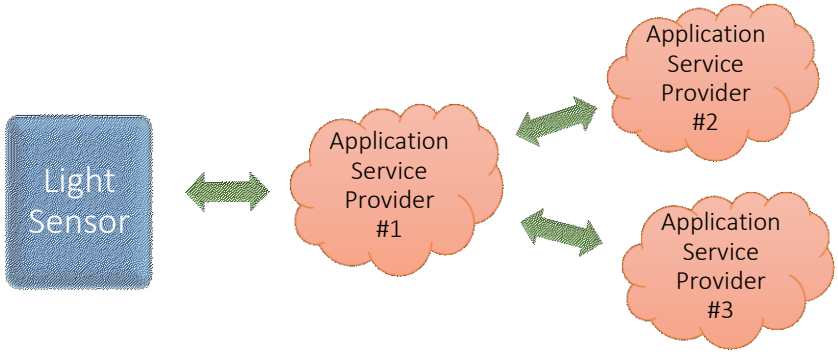


Figura 1.5: Esempio di comunicazione "*Back-End*".

1.5. Caratteristiche dei protocolli IoT

Vi sono numerosi protocolli e linguaggi adatti all' *Internet of Things*, dai tradizionali WiFi o Bluetooth ai recenti LoraWAN[24] e Sigfox[25].

Ognuno è adatto a utilizzi diversi, dipendenti da alcuni fattori chiave, quali:

- **Data Rate:** quante informazioni vengono trasmesse in un determinato lasso di tempo;
- **Power Consumption:** il consumo energetico. (Es.: I dispositivi indossabili dispongono di una riserva di energia limitata);
- **Range**: è molto importante la distanza di trasmissione la quale può infatti essere fondamentale per la scelta del protocollo utilizzato;
- **Frequency:** è molto importante anche conoscere le frequenze disponibili nell'area.

[23]**API:** Un'interfaccia di programma applicativo è un insieme di routine, protocolli e strumenti per la creazione di applicazioni software.

[24]**LoRaWAN:** Protocollo MAC (Media Access Control) per reti geografiche. È progettato per consentire ai dispositivi a bassa potenza di comunicare con le applicazioni connesse a Internet tramite connessioni wireless a lungo raggio.

[25]**Sigfox:** Operatore di rete globale francese fondato nel 2009 che costruisce reti wireless per collegare oggetti a bassa potenza come contatori elettrici e smartwatch, che devono essere costantemente accesi ed emettere piccole quantità di dati).

CAPITOLO 2:
PROTOCOLLI DI COMUNICAZIONE

Oltre alla scelta del modello di comunicazione, i dispositivi IoT per poter ricevere o inviare informazioni devono utilizzare un protocollo di comunicazione a livello applicativo.

Sono diversi i protocolli di comunicazione utilizzati e possiamo dividerli in due paradigmi:

- **Request/Response (polling)**, le informazioni vengono richieste e si rimane in attesa di una risposta, generalmente questo paradigma introduce un timer in modo da fare richieste continue per avere sempre il dato aggiornato
- **Publish/Subscribe (push-based)**, la struttura è formata da una parte che si iscrive a certi "*topic*" di cui si vogliono avere aggiornamenti, una parte che pubblica nuove informazioni per certi "*topic*", e una parte che gestisce le iscrizioni e pubblicazioni e si occupa dello scambio di messaggi. In questa tipologia quindi una volta che ci si è registrati a un certo "*topic*" si viene notificati per ogni nuova informazione, in questo modo la rete non viene saturata chiedendo continuamente aggiornamenti anche se non ci sono.

Questo capitolo introdurrà due tra i più famosi protocolli di comunicazione a livello applicativo disponibili (MQTT e CoAP) che possono essere scelti in base allo scenario, mettendo in evidenza le differenze presenti tra i due protocolli.

[1]**International Business Machines Corporation (IBM)**: Multinazionale americana di tecnologia dell'informazione con sede ad Armonk, New York, con attività in oltre 170 paesi. IBM produce e vende hardware, middleware e software per computer e fornisce servizi di hosting e consulenza in aree che vanno dai computer mainframe alle nanotecnologie.

[2]**Eurotech**: Società dedicata alla ricerca, sviluppo, produzione e commercializzazione di computer in miniatura (NanoPC) e computer ad alte prestazioni (HPC).

[3]**ISO/IEC PRF 20922**: Codice identificativo del protocollo MQTT.

[4]**TCP/IP**: sta per Transmission Control Protocol / Internet Protocol. Lo stack TCP/IP è un set completo di protocolli di rete. Il modello OSI doveva essere un modo standardizzato di connessione tra dispositivi e la maggior parte dei protocolli ha una correlazione diretta con il modello OSI.

[5]**IANA**: funzione di ICANN, una società privata americana senza scopo di lucro che sovrintende all'allocazione globale degli indirizzi IP, all'allocazione autonoma dei numeri di sistema, alla gestione della zona radice nel Domain Name System (DNS), ai tipi di media e ad altri simboli e numeri Internet relativi al protocollo Internet.

[6]**SSL**: tecnologia di sicurezza standard per stabilire un collegamento crittografato tra un server e un client, in genere un server Web (sito Web) e un browser, oppure un server di posta e un client di posta (ad esempio Outlook).

2.1. MQTT

Il "*Message Queuing Telemetry Transport*" (**MQTT**) fu inventato dal Dr. Andy Stanford-Clark di IBM[1] e Arlen Nipper di Arcom (ora Eurotech[2]) nel 1999.

È un protocollo di comunicazione "*Machine-to-Machine*" (M2M)/ *Internet of Things* (ISO/IEC PRF 20922)[3].

È stato costruito con un design "*publish/subscribe*" per il trasporto di messaggi estremamente semplice e leggero. È utile per connessioni con posizioni remote dove è necessario un codice con una breve impronta.

Ad esempio viene utilizzato nella *home automation* e in scenari di piccoli *"devices"*. È anche molto utile per applicazioni *"mobile"* grazie alle ridotte dimensioni, al basso consumo energetico, ai pacchetti di dati minimizzati e all'efficiente distribuzione di informazioni ad uno o molti ricevitori.

MQTT è un protocollo di comunicazione asincrono che lavora in cima allo *stack* TCP/IP[4].

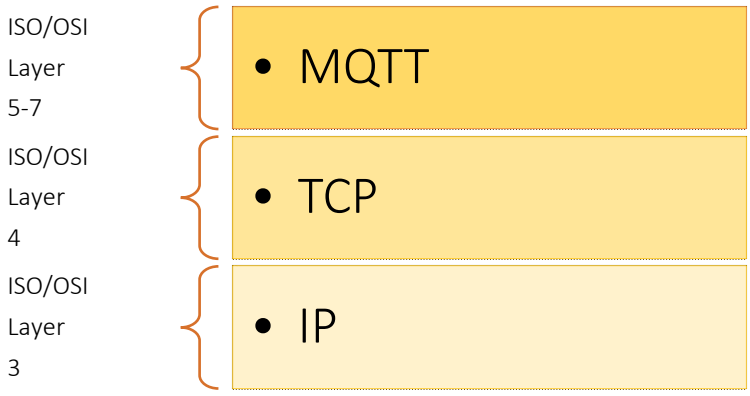


Figura 2.1: *Stack* MQTT

Dunque la porta TCP/IP 1883 è riservata con "*Internet Assigned Numbers Authority*" (**IANA**)[5] per l'utilizzo di MQTT. Anche la porta TCP/IP 8883 è registrata per l'utilizzo di MQTT con "*Secure Sockets Layer*" (**SSL**)[6].

2.1.1. Architettura

MQTT è formato da due parti:

- **Client**, che può essere sia il "*publisher*" che il "*subscriber*".
  Un client MQTT è un qualsiasi dispositivo, che sia un microcontrollore o un server, con una libreria MQTT in esecuzione e connesso ad un broker MQTT su qualsiasi tipo di rete.

Le librerie client MQTT sono disponibili per una vasta gamma di linguaggi di programmazione come ad esempio: Android, Arduino, C, C++, C#, Go, iOS, Java, Javascript, Python, .Net, ecc.;

- **Broker**, il quale è il cuore del protocollo "*publish/subscribe*".
  In base all'implementazione usata, un broker può gestire migliaia di connessioni clienti contemporaneamente.
  È il primo responsabile per la ricezione di tutti i messaggi, filtrando e decidendo in base alle iscrizioni a chi destinare i messaggi.
  Un'altra responsabilità del broker è l'autenticazione e l'autorizzazione dei client.
  Nella maggior parte dei casi è possibile estendere il broker creando autenticazioni personalizzate e integrazioni con un sistema di "*back-end*"[7].

La connessione MQTT avviene sempre tra un client e un broker (Un client non è mai connesso ad un altro client direttamente) ed è istanziata quando un client invia un messaggio CONNECT[8] al broker che deve poi rispondere con un CONNACK[9] (connect ACKNOWLEDGE) contenente lo status della connessione.

Una volta che la connessione è stata stabilita, il broker la tiene aperta per tutto il tempo fino a che il client non si disconnette o cade la connessione.

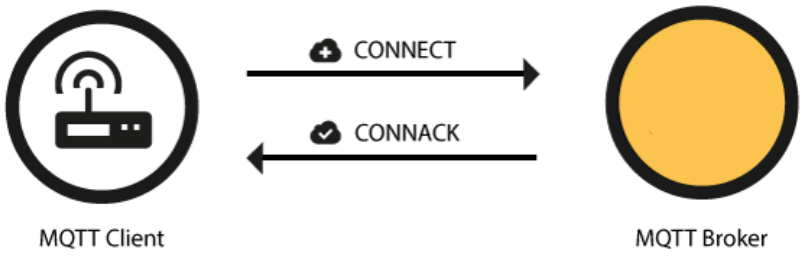


Figura 2.2: Esempio di connessione MQTT

In Figura 2.3 si può osservare un esempio di messaggio di connessione inviato dal client al broker.

[7]**Back-End**: nell'ambito del web-publishing, si indica l'interfaccia con la quale il gestore di un sito web dinamico ne gestisce i contenuti e le funzionalità. Attraverso il back-end l'amministratore del sito web può effettuare una serie di operazioni che variano a seconda della complessità e delle caratteristiche del CMS in uso.

[8]**CONNECT**: vedi Figura 2.3.

[9]**CONNACK**: vedi Figura 2.4.

[10]Return Code opzioni:

1

0x01: Connessione rifiutata, versione del protocollo inaccettabile

Il server non supporta il livello del protocollo MQTT richiesto dal client

2

0x02: Connessione rifiutata, identificatore rifiutato

L'identificatore client è UTF-8 corretto ma non consentito dal server

3

0x03: Connessione rifiutata, server non disponibile

La connessione di rete è stata stabilita ma il servizio MQTT non è disponibile

4

0x04: Connessione rifiutata, nome utente o password errati

I dati nel nome utente o password non sono corretti

5

0x05: Connessione rifiutata, non autorizzata

Il client non è autorizzato a connettersi

6 - 255

Riservato per uso futuro.

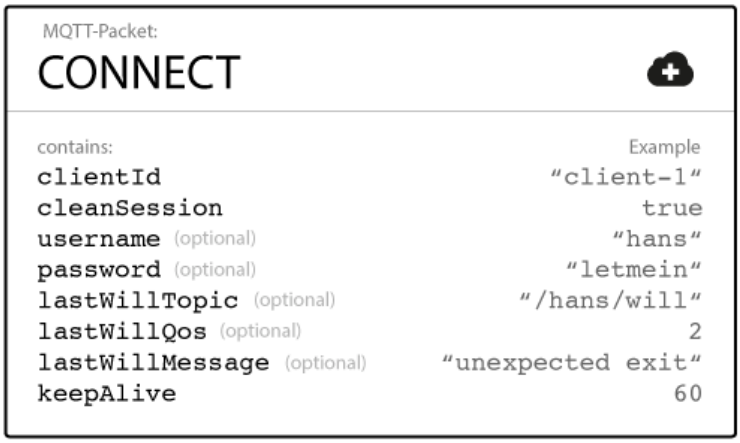


Figura 2.3: Pacchetto messaggio di CONNECT

Le opzioni in dettaglio del messaggio:

- **ClientId**, è l'identificativo del client al broker, e deve essere univoco per ogni broker;
- **CleanSession**, indica se il broker deve istanziare questo tipo di connessione con una sessione persistente oppure no. Questo vuol dire che se "*Clean Session*" è "*false"* allora è una sessione persistente e il broker salverà tutte le sottoscrizioni e i messaggi persi per questo client, altrimenti se è "*true"* il broker non salverà nessuna informazione a riguardo ma sarà uno scambio volatile di informazioni: non ne rimarrà traccia;
- **Username/Password**, sono i dati di autenticazione del client per poter aumentare la sicurezza e l'affidabilità della connessione;
- **Will Message**, è il messaggio di testamento, che permette di notificare gli altri client quando avviene una disconnessione involontaria;
- **KeepAlive**, è l'intervallo di tempo in cui il client effettua un messaggio di PING al broker e il broker risponde per tenere attiva la connessione.

Il broker nello stesso modo risponderà con un messaggio di CONNACK:

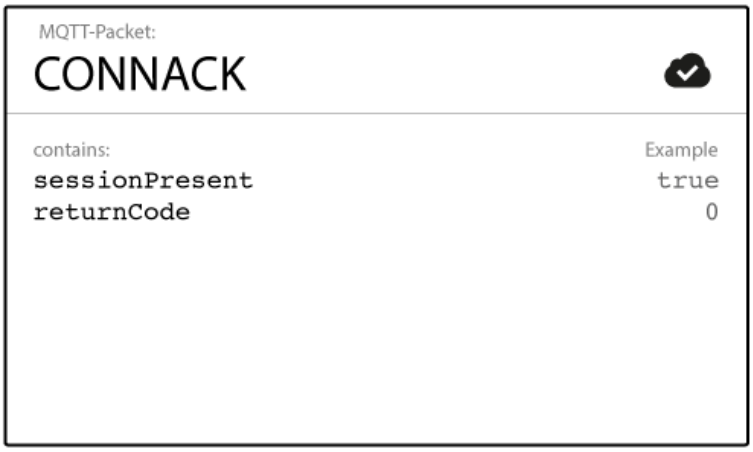


Figura 2.4: Pacchetto messaggio di CONNAK

Le opzioni in dettaglio del messaggio:

- **Session Present**, indica se il broker ha già una connessione persistente per questo client dovuta a interazioni precedenti, collegata all'opzione "*CleanSession*", vista precedentemente;
- **Return Code**, è un valore interno che può assumere un valore da 0 a 5 in base all'esito della richiesta di connessione. Ad esempio restituisce 0 se è stata accettata, e da 1 in poi se è stata rifiutata con le varie opzioni[10].

Una volta che il client MQTT si è connesso al broker può iniziare a pubblicare messaggi.

Il broker MQTT filtra i messaggi, ogni messaggio deve contenere il "*topic*", che sarà utilizzato dal broker per poter indirizzare il messaggio ai client interessati. Ogni messaggio ha un "*payload*"[11] che contiene l'informazione che si intende spedire, in formato byte. MQTT è un protocollo di tipo "*data-agnostic*"[12].

Di seguito è riportato il messaggio di PUBLISH:

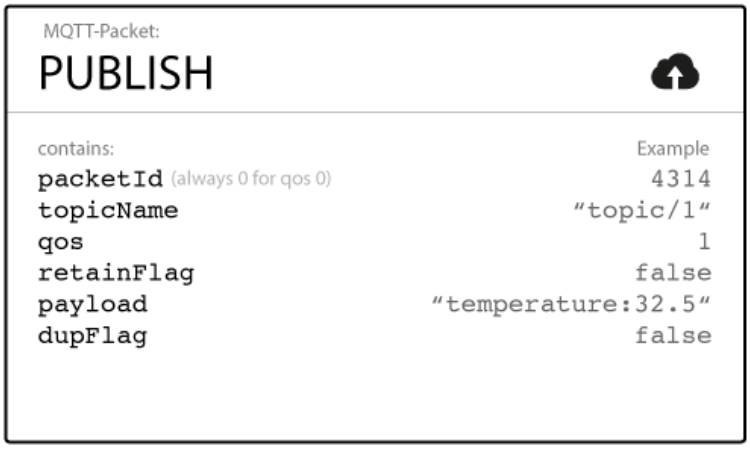


Figura 2.5: Pacchetto messaggio di PUBLISH

Le opzioni in dettaglio sono:

- **Packet Identifier (PacketID)**, è un identificatore unico tra il client e il broker per identificare il messaggio;
- **TopicName**, una stringa che indica la gerarchia del "*topic*": gli slash vengono utilizzati come delimitatori;
- **QoS** (*Quality of Service*), garantisce l'affidabilità del servizio e può essere di tre tipi:
  0: "*Fire and forget*", il messaggio viene inviato una volta sola senza nessuna conferma di ricezione;

[11]**Payload**: vedi Figura 2.5

[12]**Data agnostic**: Nella tecnologia dell'informazione, "agnostico" si riferisce alla capacità di un sistema di funzionare con molti sistemi, piuttosto che essere progettato per un singolo sistema.
Un sistema indipendente dai dati può funzionare con informazioni ricevute da database eterogenei, ovvero database con formati di dati diversi. Laddove manca un protocollo comune, spetta al sistema utilizzare i dati per essere in grado di interpretare ciascun formato di dati in entrata in uno che può utilizzare. Nel caso del protocollo MQTT, all'invio del messaggio sarà una scelta del mittente se il dato inviato risulterà essere in binary-data, textual-data, XML o JSON.

[13]**Four-way Handshake**: processo di scambio di 4 messaggi tra un punto di accesso e il dispositivo client per generare alcune chiavi di crittografia che possono essere utilizzate per crittografare i dati effettivi inviati tramite supporto wireless.

1: "*Delivered at least once*", il messaggio viene inviato almeno una volta e richiede una conferma di recezione (ACKNOLEDGE);
2: "*Delivery exactly once*", viene utilizzato il meccanismo "*four-way handshake*"[13] per assicurarsi che il messaggio sia stato consegnato esattamente una volta, risulta essere il più lento ma anche il più sicuro.

- **Retain-Flag**, indica se il messaggio deve essere salvato dal broker per questo "*topic*", come ultimo valore conosciuto.
  In questo modo se un client si iscrive a un "*topic*" riceve subito l'ultima informazione disponibile;
- **Payload**, è il contenuto vero e proprio del messaggio;
- **Dup Flag**, indica che il messaggio in questione è un duplicato poiché per l'originale non è stato ricevuto un messaggio di conferma.
  Quindi quando un client pubblica un messaggio il broker lo riceve e lo spedisce alle parti interessate che sono iscritte a quel determinato "*topic*".
  Il client che ha inizialmente pubblicato il messaggio non deve più occuparsi di nulla perché da lì in poi è il broker che si occuperà del messaggio.

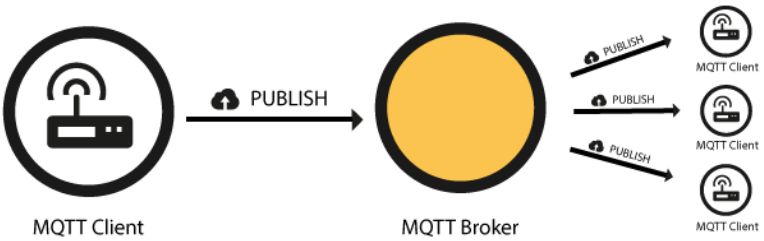


Figura 2.6: Esempio di pubblicazione di un messaggio

Allo stesso modo un client può iscriversi a un determinato "*topic*", tramite il comando SUBSCRIBE (le informazioni del pacchetto sono mostrate in Figura 2.7).

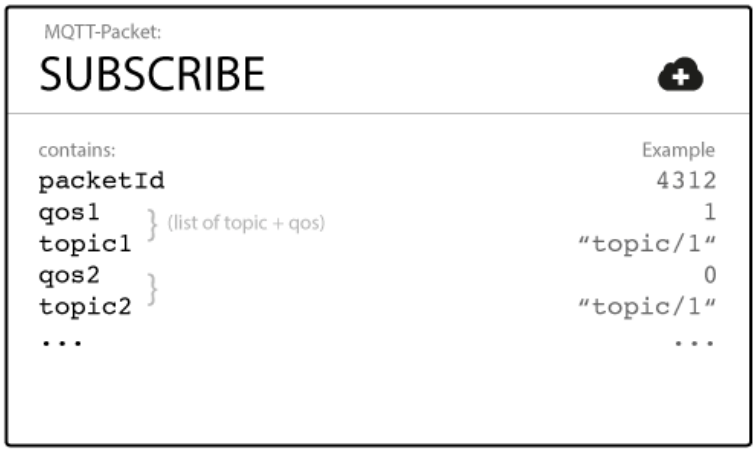


Figura 2.7: Pacchetto messaggio di SUBSCRIBE.

Le opzioni in dettaglio del pacchetto sono:

- **Packet Identifier (PacketID)**, è un identificativo univoco per identificare il messaggio (esattamente come per il PUBLISH);
- **List of Subscription**, un messaggio di SUBSCRIBE può contenere un numero arbitrario di sottoscrizioni per un client. Ogni sottoscrizione è una coppia formata da "*topic*" e "*QoS level*". Il "*topic*" può anche contenere una wildcard[14] che rende possibile iscrivere per più "*topic*" di un certo tipo.

Oltre a questi messaggi principali esistono anche altri messaggi, ad esempio: per confermare, per annullare e confermare l'annullamento della sottoscrizione.

In MQTT gli argomenti sono gerarchici come in un sistema di archiviazione (Es.: Cucina/forno/temperatura).
I caratteri jolly sono consentiti durante la registrazione di un abbonamento (ma non durante la pubblicazione), consentendo ai client di osservare intere gerarchie.

A differenza di una coda di messaggi ("*message queue*"[15]), i broker MQTT non consentono il backup dei messaggi persistenti all'interno del server.

Un esempio di comunicazione con un generico sensore IoT è il seguente:

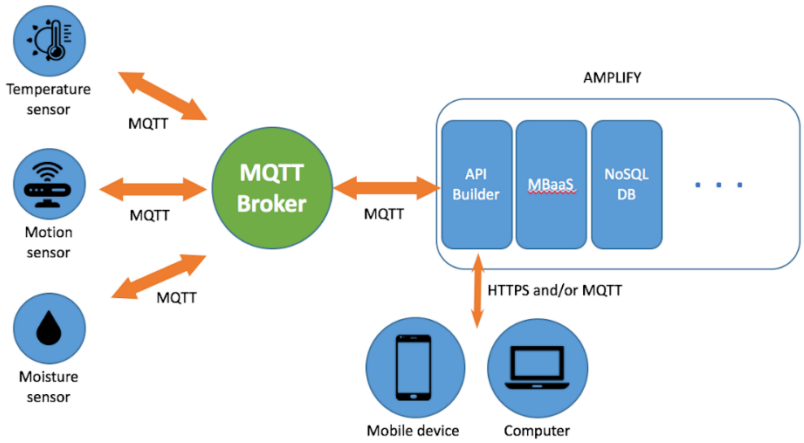


Figura 2.8: Esempio di comunicazione MQTT.

[14]**Wildcards**:

Segno di somma (+): è un carattere jolly a livello singolo che associa ogni nome per uno specifico livello di "*topic*". È possibile utilizzare questo carattere jolly anziché specificare un nome per ogni livello di "*topic*" nel filtro argomenti.

Hashtag (#): è un jolly multi-livello che può essere utilizzato solo alla fine del filtro argomenti (come ultimo livello) e corrisponde a qualsiasi "*topic*" i cui livelli precedenti corrispondono all'argomento specificati sul lato sinistro del simbolo.

[15]**Message queue**: Coda di messaggi inviati tra le applicazioni. Include una sequenza di oggetti di lavoro in attesa di essere elaborati.

2.1.2. Sicurezza

Per la connessione, i broker MQTT possono richiedere un'autenticazione tramite "*Username/Password*" al client (dalla V3.1 del protocollo in poi).
Per assicurare la privacy, la connessione TCP può essere criptata con SSL o TLS[16] con la comunicazione dalla porta 8883, indipendentemente dal protocollo MQTT stesso. Una sicurezza aggiuntiva può essere fornita da un'applicazione che codifica i dati che invia e riceve, la quale però non è integrata nel protocollo MQTT, al fine di mantenerlo semplice e leggero.

2.1.3. MQTT-SN

Sebbene MQTT sia progettato per essere leggero presenta due inconvenienti per dispositivi molto limitati.

Ogni client MQTT deve supportare TCP e in genere mantiene sempre aperta una connessione al broker.
Questo, per alcuni ambienti in cui la perdita di pacchetti è elevata o le risorse di elaborazione scarseggiano, è un problema.

I nomi dei "*topic*" del protocollo MQTT sono spesso stringhe lunghe, il che li rende poco pratici per lo standard 802.15.4.[17].

Entrambi questi problemi vengono risolti dal protocollo MQTT-SN, il quale definisce una mappatura "*User Data Protocol*" (UDP)[18] ed aggiunge un supporto al broker per l'indicizzazione del nome dei "*topics*".

[16]TLS: Il protocollo TLS consente alle applicazioni client/server di comunicare attraverso una rete in modo tale da prevenire il "tampering" (manomissione) dei dati, la falsificazione e l'intercettazione. È un protocollo standard IETF che, nella sua ultima versione, è definito nella RFC 5246.

[17]Standard 802.15.4.: Concepito per regolamentare il livello fisico e il livello MAC (Media Access Control) di reti in area personale che lavorano con basse velocità di trasferimento dati.

[18]UDP: Protocollo di livello di trasporto OSI (Open Systems Interconnection) per applicazioni di rete client-server. UDP utilizza un semplice modello di trasmissione ma non utilizza dialoghi di handshaking per affidabilità, ordinazione e integrità dei dati. Il protocollo presuppone che il controllo degli errori e la correzione non siano necessari, evitando così l'elaborazione a livello di interfaccia di rete.

2.2. CoAP

Uno degli obiettivi principali del "*Constrained Application Protocol*" (**CoAP**) è realizzare un protocollo WEB adatto alle esigenze di dispositivi con risorse limitate in termini computazionali ed energetiche, come definito in **RFC 7275**[19].

CoAP permette a questi "*constrained devices*", denominati nodi ("*nodes*"), di comunicare con la vastità dell'Internet.
I nodi solitamente sono composti da un microcontrollore a 8-bit con una quantità limitata di ROM e RAM, mentre i "*constrained networks*" sono solitamente IPv6 con "*Low-Power Wireless Personal Area Networks*" (6LoWPANs) e hanno solitamente un grande tasso di perdita di informazioni e un "*throughput*"[20] tipico di 10 kbit/s.

Il protocollo CoAP è anche utilizzato attraverso altri meccanismi come, ad esempio, SMS in comunicazioni "*mobile*" con i network.

Il modo con cui si vuole realizzare questo protocollo non consiste in una semplice compressione del protocollo HTTP[21], quanto piuttosto nell'implementazione di un sottoinsieme delle funzionalità offerte dalla architettura "*Representational state transfer*" (**REST**), in comune con HTTP, ottimizzandole nell'ottica delle applicazioni di comunicazione M2M (Es.: *smart energy*, *building automation*).
Il protocollo CoAP offre un modello di interazione "*request/response*" tra gli "*application endpoint*". Include, inoltre, un supporto integrato per la ricerca dei servizi e delle risorse e concetti chiave del WEB come URLs e "*Internet Media Types*".

Il lavoro di standardizzazione del protocollo è opera del "*Constrained RESTful Environments* (**CoRE**) *Working Group*" della *Internet Engineering Task Force* (**IETF**)[22].

2.2.1. Architettura

Alcuni aspetti dell'architettura di funzionamento di CoAP è possibile ritrovarli all'interno del protocollo HTTP, come ad esempio la struttura client/server.
Una richiesta CoAP è equivalente ad una HTTP ed è inviata da un client CoAP verso un server CoAP per richiedere un'azione (tramite la specifica di un metodo) su una risorsa identificata da una URI[23].

[19] **RFC 7275**: Codice identificativo del documento "Inter-Chassis Communication Protocol for Layer 2 Virtual Private Network (L2VPN) Provider Edge (PE) Redundancy"

[20] **Troughput**: Capacità effettiva di un canale di telecomunicazione, è la capacità di trasmissione che viene usata, ed è minore di quella teorica.

[21] **HTTP**: Protocollo di trasferimento *HyperText*. HTTP è il protocollo utilizzato dal World Wide Web che definisce il modo in cui i messaggi vengono formattati e trasmessi e quali azioni devono eseguire i server Web e i browser in risposta a vari comandi.

[22] **IEFT**: Organizzazione di standard aperti, che sviluppa e promuove standard Internet volontari, in particolare gli standard che comprendono la suite di protocolli Internet (TCP/IP). Non ha un elenco di iscritti formale o requisiti di iscrizione.

[23] **URI**: In informatica è una sequenza di caratteri che identifica universalmente ed univocamente una risorsa. (Es.: un indirizzo web (**URL**), un documento, un'immagine, un file, ecc.)

Come HTTP, il CoAP è un protocollo di trasferimento di documenti ma a differenza di HTTP è progettato per i bisogni dei nodi.

CoAP è progettato per interagire con HTTP e il Web RESTful tramite semplici proxy[24] ed è basato su datagramma[25].

Gli scambi di messaggi sono asincroni e vengono eseguiti tramite il protocollo di livello di trasporto UDP.
Le connessioni UDP, inoltre, abilitano un risveglio rapido della connessione e permettono ai dispositivi di rimanere in uno stato di *sleep* per lunghi periodi in modo da risparmiare batteria.

I pacchetti CoAP sono molto più piccoli dei flussi TCP HTTP.
Per risparmiare spazio vengono utilizzati campi di bit e "*mapping*" da stringhe a numeri interi.
I pacchetti sono semplici da generare e possono essere analizzati sul posto senza consumare ulteriore RAM nei nodi.

Il protocollo CoAP è stato progettato per fornire un'integrazione trasparente con il Web.
Un *cross-proxy* è un elemento della rete che fornisce funzionalità di "*protocol translation*"[26] per permettere l'integrazione tra endpoint HTTP e CoAP.

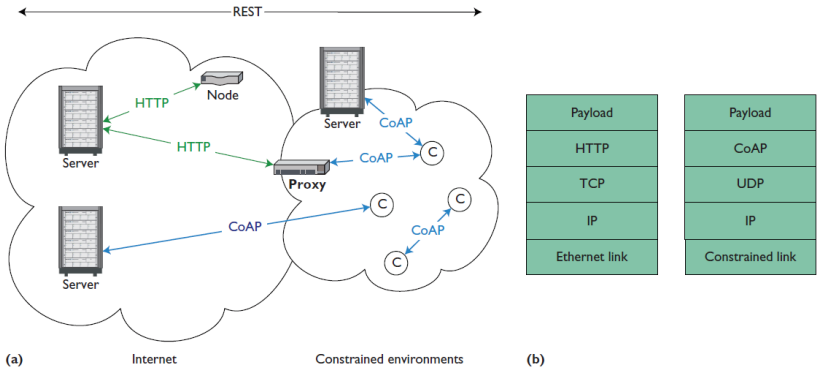


Figura 2.9: CoAP cross-proxy

CoAP è basato su UDP e l'omissione di TCP consente l'utilizzo completo della rete IP nei microcontrollori.

CoAP estende il modello di richiesta HTTP con la capacità di osservare una risorsa. Quando il flag di osservazione è impostato su una richiesta GAP CoAP, il server può continuare a rispondere dopo il trasferimento del documento iniziale.
Ciò consente ai server di trasmettere in streaming le modifiche di stato ai client non appena si verificano, entrambe le estremità possono annullare l'osservazione.

[24]**Proxy**: Nelle reti di computer, un server proxy è un server (un sistema informatico o un'applicazione) che funge da intermediario per le richieste dei clienti che cercano risorse da altri server.

[25]**Datagramma**: In informatica, in una rete di comunicazione a commutazione di pacchetto, è pacchetto dati di dimensioni limitate contenente gli indirizzi di provenienza e di destinazione del pacchetto e l'indicazione delle risorse e servizi utilizzati; la spedizione del pacchetto non richiede quindi scambi preventivi di messaggi tra i data terminal equipment della sorgente e della destinazione del pacchetto.

[26]**Protocol translation**: Dispositivo utilizzato per convertire il protocollo standard o proprietario di un dispositivo nel protocollo adatto all'altro dispositivo o strumenti per raggiungere l'interoperabilità.

CoAP definisce un meccanismo standard per il rilevamento delle risorse. I server forniscono un elenco delle loro risorse (insieme ai metadati[27] su di loro) in "*/well-known/core*".
Questi collegamenti si trovano nel tipo di supporto in formato applicazione/collegamento e consentono a un client di scoprire quali risorse vengono fornite e quali tipi di supporto sono.

Il protocollo CoAP ha dovuto colmare alcuni vuoti dovuti dalla natura dell'UDP integrando alcuni meccanismi per raggiungere l'affidabilità.
I messaggi di richiesta e risposta possono essere marcati come "*confirmable*" o "*non-confirmable*".
I messaggi di tipo "*confirmable*" devono essere riconosciuti ed accettati dal ricevitore tramite un invio di un ACK di conferma, mentre i messaggi "*non-confirmable*" sono messaggi "*fire and forget*".

Due byte nell'intestazione di ogni pacchetto indicano il tipo di messaggio e il livello di QoS richiesto.

Vi sono quattro tipi di messaggi:

- **Confirmable**, un messaggio di richiesta che necessita di un messaggio di conferma (ACK).
  La risposta può essere spedita sia in maniera sincrona quindi insieme all'ACK che, se ha bisogno di più tempo di computazione, in un secondo momento in maniera asincrona con un messaggio separato.
- **Non-Confirmable**, un messaggio che non ha bisogno di un ACK di conferma.
- **Acknowledgement**, conferma la ricezione di un messaggio "*Confirmable*".
- **Reset**, un messaggio di reset viene inviato a seguito di una richiesta che il server non è in grado di processare in quanto mancante del contesto.

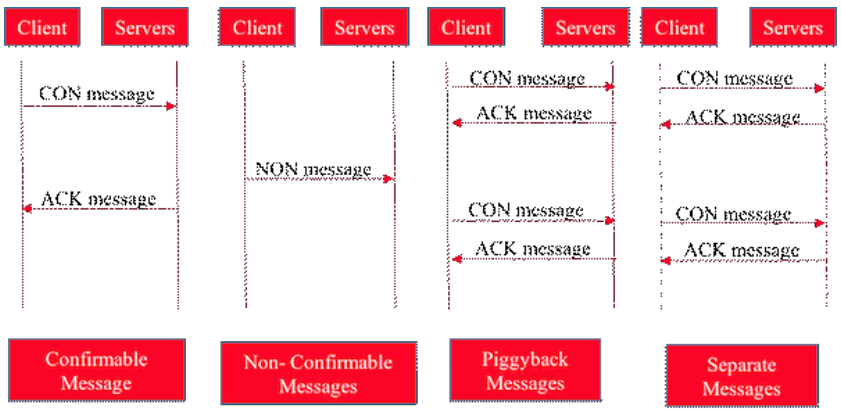


Figura 2.11: Esempio di messaggi CoAP

[27]**Metadata**: In informatica il metadato è un sistema strutturato di dati sui dati. Il suo scopo è di descrivere il contenuto, la struttura e l'ambito in cui s'inquadra un documento informatico, per la sua gestione e archiviazione nel tempo.

Come HTTP, CoAP supporta la negoziazione dei contenuti.
I client utilizzano le opzioni *Accept* per esprimere una rappresentazione preferita di una risorsa e i server rispondono con un'opzione *Content-Type* per riferire ai client cosa stanno ricevendo.
Come per HTTP, ciò consente a client e server di evolversi in modo indipendente, aggiungendo nuove rappresentazioni senza influenzarsi a vicenda.

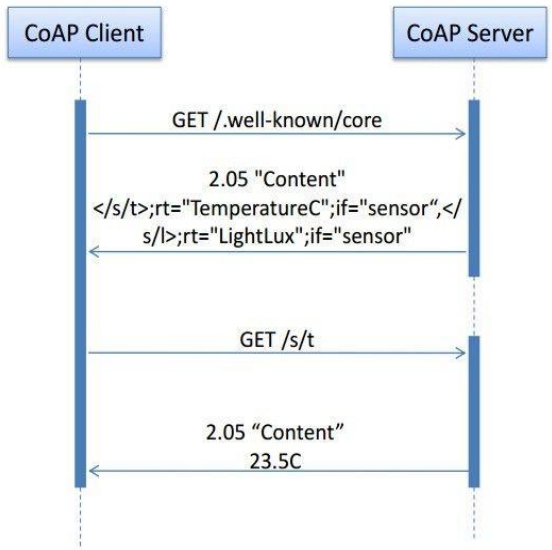


Figura 2.12: Esempio di comunicazione CoAP

2.2.2. Sicurezza

Il protocollo CoAP è stato creato per comunicazioni IoT e M2M e non include al suo interno nessuna caratteristica di sicurezza.

Poiché CoAP è basato su UDP e non su TCP, SSL/TLS non sono disponibili per fornire sicurezza.
Il "*Datagram Transport Layer Security*" (**DTLS**)[28] offre le stesse garanzie di TLS, ossia: fornisce autenticazione, integrità dei dati, gestione automatica delle chiavi e algoritmi di criptazione, tutto questo per i trasferimenti di dati su UDP.

In genere, i dispositivi CoAP compatibili con DTLS supporteranno RSA[29] e AES[30] o ECC[31] e AES.

DLTS, però, non è stato pensato per l'IoT, perché non supporta il multi-cast[32] che è uno dei primi vantaggi di CoAP comparato con altri protocolli di comunicazione.

[28]**DTLS**: Protocollo di comunicazione che fornisce sicurezza per le applicazioni basate su datagramma consentendo loro di comunicare in modo progettato per prevenire intercettazioni, manomissioni o contraffazione di messaggi.

[29]**RSA**: Uno dei primi cryptosystems a chiave pubblica ed è ampiamente utilizzato per la trasmissione sicura dei dati. In un tale sistema crittografico, la chiave di crittografia è pubblica ed è diversa dalla chiave di decrittazione che è mantenuta segreta (privata).

[30]**AES**: In crittografia, l'Advanced Encryption Standard (AES), conosciuto anche come Rijndael, è un algoritmo di cifratura a schemi usati come standard dal governo degli Stati Uniti d'America

[31]**ECC**: Codice di 288 bytes per il rilevamento e la correzione degli errori, da parte del computer, durante la fase di lettura dei dati.

[32]**Multi-cast**: Nella rete di computer, il multicast è la comunicazione di gruppo in cui la trasmissione dei dati è indirizzata a un gruppo di computer di destinazione contemporaneamente.

L'handshake del DTLS richiede pacchetti addizionali che incrementano il traffico di rete e richiedono maggior computazione, così facendo consumano anche più batteria per dispositivi che avrebbero bisogno il minor consumo possibile.
Inoltre anche se è HTML-compatibile, CoAP con DTLS può creare una confusione addizionale ai server HTTP a causa della diversa struttura dei pacchetti.

2.2.3. Problemi di NAT

In CoAP, un nodo sensore è in genere un server, non un client (sebbene possa essere entrambi).
Il sensore (o attuatore) fornisce risorse a cui i *client* possono accedere per leggere o modificare lo stato del sensore.

Poiché i sensori CoAP sono server, devono essere in grado di ricevere pacchetti in entrata.

Per funzionare correttamente dietro NAT[33], un dispositivo può prima inviare una richiesta al server, come avviene in LWM2M ("*Lightweight Machine to Machine*"), consentendo al router di associare i due.
Sebbene CoAP non richieda IPv6, è più semplice da utilizzare in ambienti IP in cui i dispositivi sono direttamente instradabili.

[33]NAT: Nel campo delle reti telematiche, il network address translation o NAT, è conosciuto anche come network masquerading.
Si tratta di un metodo per rimappare uno spazio di indirizzi IP in un altro modificando le informazioni sull'indirizzo di rete nell'intestazione IP dei pacchetti mentre sono in transito attraverso un dispositivo di instradamento del traffico.
La tecnica era originariamente utilizzata come scorciatoia per evitare la necessità di reindirizzare tutti gli host quando veniva spostata una rete.
È diventato uno strumento popolare ed essenziale per conservare lo spazio degli indirizzi globale di fronte all'esaurimento degli indirizzi IPv4.

2.3. Confronto tra MQTT e CoAP

MQTT e CoAP sono entrambi utili come protocolli IoT, ma presentano differenze fondamentali.

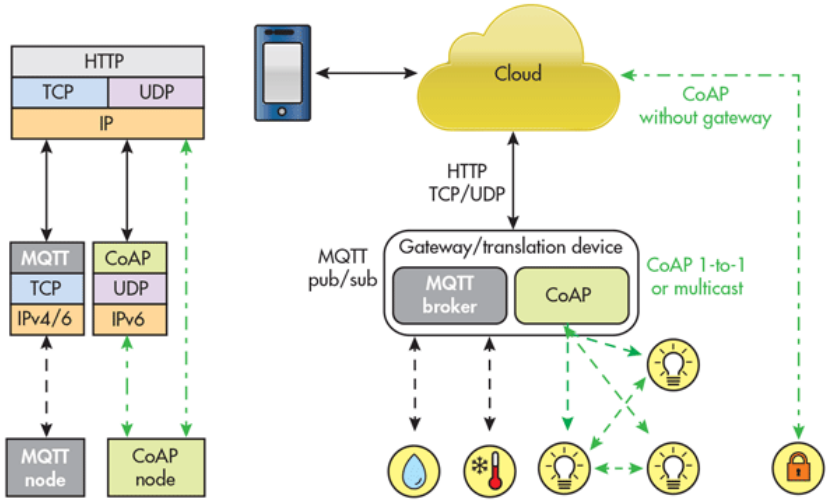


Figura 2.13: Differenze tra MQTT e CoAP

MQTT è un protocollo di comunicazione "molti-a-molti" per il passaggio di messaggi tra più client attraverso un broker centrale. Disaccoppia il produttore e il consumatore consentendo ai client di pubblicare e far decidere al broker dove instradare e copiare i messaggi.
Pur possedendo un po' di supporto per la persistenza, funziona meglio come bus di comunicazione per dati live.

CoAP è principalmente un protocollo "uno-ad-uno" per il trasferimento di informazioni sullo stato tra client e server.
Sebbene abbia il supporto per l'osservazione delle risorse, CoAP è più adatto a un modello di trasferimento dello stato, non puramente basato sugli eventi.

I client MQTT stabiliscono una connessione TCP in uscita di lunga durata con un broker. Questo di solito non presenta alcun problema per i dispositivi dietro NAT.
I client e i server CoAP inviano e ricevono pacchetti UDP. In ambienti NAT, il tunneling[34] o il port forwarding[35] possono essere utilizzati per consentire CoAP, oppure i dispositivi possono prima avviare una connessione alla testata come in LWM2M.

MQTT non fornisce supporto per etichettare i messaggi con tipi o altri metadati per aiutare i client a capirlo. I messaggi MQTT possono essere utilizzati per qualsiasi scopo, ma tutti i client devono conoscere i formati dei messaggi in anticipo per consentire la comunicazione.

[34]**Tunneling**: Nelle reti di computer, un protocollo di tunneling è un protocollo di comunicazione che consente lo spostamento dei dati da una rete all'altra. Implica il consentire l'invio di comunicazioni di rete private attraverso una rete pubblica (come Internet) attraverso un processo chiamato incapsulamento. Poiché il tunneling comporta il riconfezionamento dei dati sul traffico in una forma diversa, magari con la crittografia di serie, può nascondere la natura del traffico che viene eseguito attraverso un tunnel.

[35]**Port forwarding**: Nella rete di computer, il port forwarding o il mapping delle porte è un'applicazione di NAT (Network Address Translation) che reindirizza una richiesta di comunicazione da un indirizzo e una combinazione di numeri di porta a un altro mentre i pacchetti attraversano un gateway di rete, come un router o un firewall.

Al contrario, CoAP fornisce supporto integrato per la negoziazione e il rilevamento dei contenuti, consentendo ai dispositivi di sondarsi a vicenda per trovare il modo di scambiare dati.

Entrambi i protocolli hanno pro e contro, la scelta di quello giusto dipende dalla propria applicazione.

CAPITOLO 3:
PIATTAFORME IOT

Tra le caratteristiche che deve avere una piattaforma IoT, quella di basarsi su un corretto posizionamento delle risorse di calcolo nella rete è certamente fondamentale per tradurre l' "*Internet of Things*" in valore di business. È infatti spesso necessario rendere più rapidi i processi di elaborazione e analisi dei dati abbracciando le logiche dell' "*edge computing*"[1]

Come definire questo "corretto posizionamento"? Che elementi vanno tenuti in considerazione?

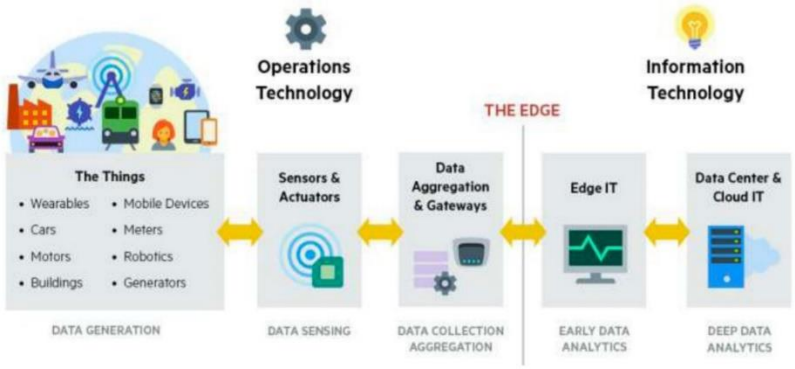


Figura 3.1: Internet of Things: il margine del network

Questo capitolo tratterà delle piattaforme IoT, cosa sono, come possono essere utilizzate e cosa possono apportare in campo aziendale, e darà un particolare sguardo ad alcune delle piattaforme presenti sul mercato per arrivare alla piattaforma scelta per lo sviluppo dell'applicazione relativa al progetto.

[1]**Edge-computing**: Paradigma di calcolo distribuito che avvicina il calcolo e l'archiviazione dei dati alla posizione necessaria, per migliorare i tempi di risposta e risparmiare larghezza di banda.
Le origini dell'edge-computing risiedono nelle reti di distribuzione dei contenuti create alla fine degli anni '90 per servire contenuti web e video da server perimetrali distribuiti vicino agli utenti.
All'inizio degli anni 2000, queste reti si sono evolute per ospitare applicazioni e componenti applicativi sui server perimetrali, dando origine ai primi servizi di edge-computing commerciale.
L'edge-computing moderno estende significativamente questo approccio attraverso la tecnologia di virtualizzazione che semplifica l'implementazione e l'esecuzione di una gamma più ampia di applicazioni sui server perimetrali.

3.1. Cos'è una piattaforma IoT

Una piattaforma IoT è una tecnologia multistrato che consente il provisioning[2], la gestione e l'automazione semplici dei dispositivi connessi nell'universo di "*Internet of Things*".

Fondamentalmente collega l'hardware, per quanto diversificato, al cloud utilizzando opzioni di connettività flessibili, meccanismi di sicurezza di livello aziendale e ampi poteri di elaborazione dei dati.

Per gli sviluppatori, una piattaforma IoT offre una serie di funzionalità pronte per l'uso che accelerano notevolmente lo sviluppo di applicazioni per dispositivi connessi e si occupano della scalabilità e della compatibilità tra dispositivi.

Pertanto, una piattaforma IoT può "indossare cappelli diversi" a seconda di come la guardi.
Viene comunemente chiamato middleware quando parliamo di come collega i dispositivi remoti alle applicazioni dell'utente (o altri dispositivi) e gestisce tutte le interazioni tra l'hardware e i livelli dell'applicazione.
È anche nota come piattaforma di abilitazione cloud o piattaforma di abilitazione IoT per individuare il suo valore commerciale principale, ovvero potenziare i dispositivi standard con applicazioni e servizi basati su cloud.
Infine, sotto il nome della piattaforma di abilitazione delle applicazioni IoT, sposta l'attenzione su uno strumento chiave per gli sviluppatori IoT.

3.1.1. Piattaforma IoT come middleware

Le piattaforme IoT sono nate sotto forma di middleware IoT, il cui scopo era quello di fungere da intermediario tra i livelli hardware e applicativo. Le sue attività principali includevano la raccolta di dati dai dispositivi tramite protocolli e topologie di rete diversi, configurazione e controllo dei dispositivi remoti, gestione dei dispositivi e aggiornamenti del firmware via "*over-the-air*"[3].

Per essere utilizzato negli ecosistemi IoT eterogenei della vita reale, il middleware IoT dovrebbe supportare l'integrazione con quasi tutti i dispositivi connessi e integrarsi con le applicazioni di terze parti utilizzate dal dispositivo.

[2]**Provisioning**: Nel settore delle telecomunicazioni, il provisioning prevede il processo di preparazione e equipaggiamento di una rete per consentirle di fornire nuovi servizi ai propri utenti.

[3]**Over-the-air**: si riferisce a vari metodi di distribuzione di nuovo software, impostazioni di configurazione e persino aggiornamento delle chiavi di crittografia a dispositivi come telefoni cellulari, set-top box o apparecchiature di comunicazione vocale sicure (radio crittografate a 2 vie).
Una caratteristica importante di OTA è che una posizione centrale può inviare un aggiornamento a tutti gli utenti, che non sono in grado di rifiutare, sconfiggere o alterare, e che l'aggiornamento si applica immediatamente a tutti gli utenti del canale.

[4]**Front-end**: Lo sviluppo Web front-end, noto anche come sviluppo lato client, è la pratica di produrre HTML, CSS e JavaScript per un sito Web o un'applicazione Web in modo che un utente possa vederli e interagire direttamente con essi.
La sfida associata allo sviluppo del front-end è che gli strumenti e le tecniche utilizzati per creare il front-end di un sito Web cambiano costantemente e quindi lo sviluppatore deve essere costantemente consapevole di come si sta sviluppando il campo.

Questa indipendenza dall'hardware sottostante e dal software a strapiombo consente a un'unica piattaforma IoT di gestire qualsiasi tipo di dispositivo connesso nello stesso modo.

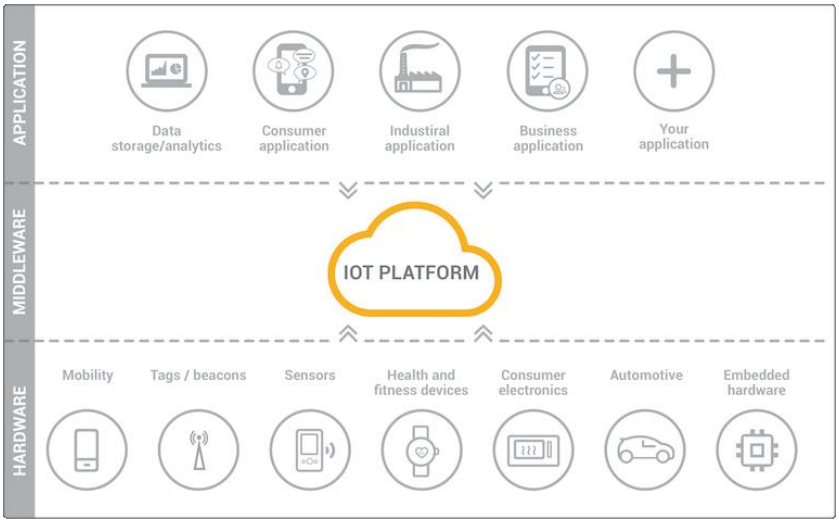


Figura 3.2: Piattaforma IoT come Middelware

Le moderne piattaforme IoT vanno oltre e introducono una varietà di preziose funzionalità anche a livello di hardware e applicazioni.
Forniscono componenti per "*front-end*"[4] e analisi, elaborazione dei dati su dispositivo e distribuzione basata su cloud.
Alcuni di loro possono gestire l'implementazione della soluzione IoT "*end-to-end*" da zero.

3.1.2. Stack tecnologico della piattaforma IoT

Nei quattro livelli tipici dello *stack* IoT, ovvero: elementi, connettività, funzionalità IoT di base e applicazioni e analisi; una piattaforma IoT di fascia alta dovrebbe fornire la maggior parte delle funzionalità IoT necessarie per lo sviluppo dei dispositivi connessi e cose "*smart*".

I dispositivi si connettono alla piattaforma, che si trova nel cloud o nel data center locale, direttamente o utilizzando un gateway IoT.
Un gateway risulta utile quando i tuoi endpoint non sono in grado di comunicare direttamente con il cloud o, ad esempio, se si ha bisogno di un po' di potenza di calcolo al limite ("*edge*").

È inoltre possibile utilizzare un gateway IoT per convertire i protocolli, ad esempio quando gli endpoint si trovano nella rete LoRaWan ma sono necessari per comunicare con il cloud su MQTT.

Una piattaforma IoT stessa può essere scomposta in più livelli:

- In fondo c'è il livello dell'infrastruttura, che consente il funzionamento della piattaforma.
  Qui si possono trovare componenti per la gestione dei container, la messaggistica della piattaforma interna, l'orchestrazione dei cluster di soluzioni IoT e altri.
- Il livello di comunicazione abilita la messaggistica per i dispositivi; in altre parole, è qui che i dispositivi si connettono al cloud per eseguire diverse operazioni.
- Il livello seguente rappresenta le principali funzionalità IoT fornite dalla piattaforma.
  Tra quelli essenziali ci sono la raccolta dei dati, la gestione dei dispositivi, la gestione della configurazione, la messaggistica e gli aggiornamenti del software OTA.

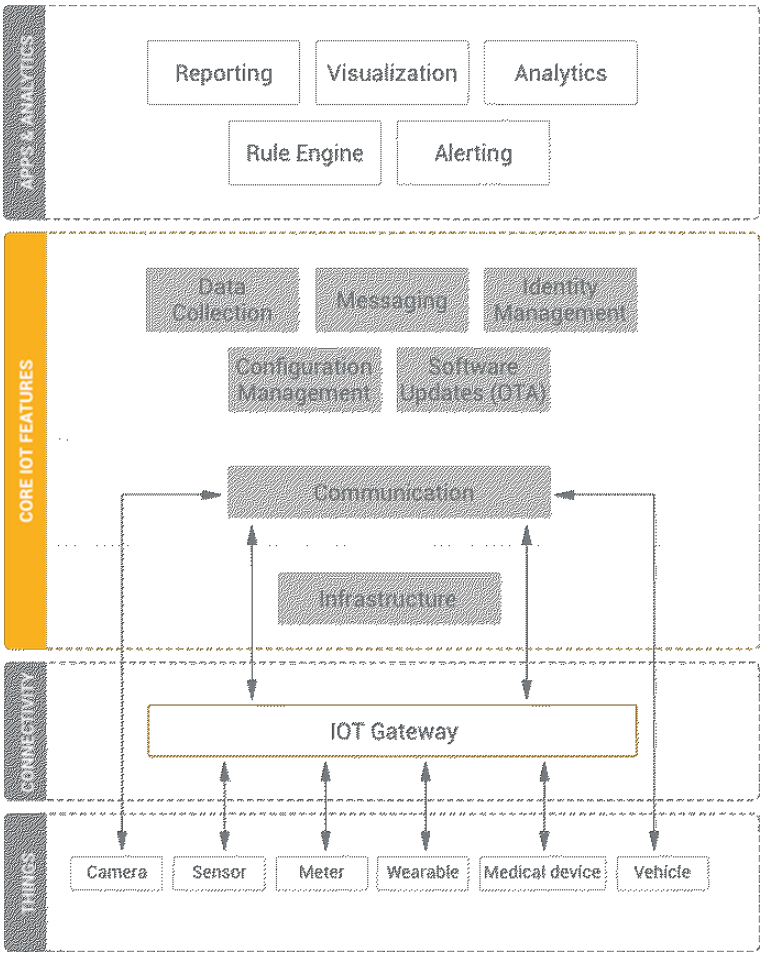


Figura 3.3: Stack delle piattaforme IoT

Oltre alle funzionalità IoT di base, esiste un altro livello, meno correlato allo scambio di dati tra dispositivi ma piuttosto all'elaborazione di questi dati nella piattaforma.

È presente un report che consente di generare report personalizzati. Esiste una visualizzazione per la rappresentazione dei dati nelle applicazioni utente. Quindi, sono disponibili un motore di regole, analisi e avvisi per la notifica di eventuali anomalie rilevate nella soluzione IoT.

È importante sottolineare che le migliori piattaforme IoT consentono di aggiungere i propri componenti specifici del settore e applicazioni di terze parti.
Senza tale flessibilità, l'adattamento di una piattaforma IoT per un particolare scenario aziendale potrebbe comportare costi aggiuntivi significativi e ritardare la consegna della soluzione a tempo indeterminato.

3.1.3. Piattaforme IoT avanzate

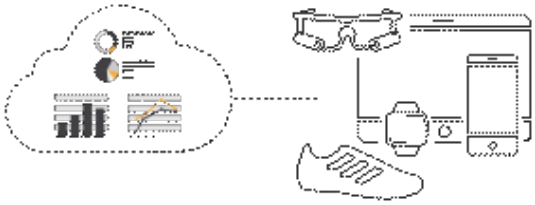
Esistono altri criteri importanti che differenziano le piattaforme IoT tra loro, come la scalabilità, la personalizzazione, la facilità d'uso, il controllo del codice, l'integrazione con software di terze parti, le opzioni di distribuzione e il livello di sicurezza dei dati.

- Scalabile (cloud nativo): piattaforme IoT avanzate assicurano una scalabilità elastica su qualsiasi numero di endpoint richiesti dal client.
  Questa funzionalità è data per scontata per le distribuzioni di cloud pubblico, ma dovrebbe essere messa alla prova specificamente in caso di una distribuzione locale, comprese le capacità di bilanciamento del carico della piattaforma per massimizzare le prestazioni del cluster di server;
- Personalizzabile: un fattore cruciale per la velocità di consegna.
  Si riferisce strettamente alla flessibilità delle API di integrazione, all'accoppiamento non corretto dei componenti della piattaforma e alla trasparenza del codice sorgente.
  Per soluzioni IoT su piccola scala e poco impegnative, possono essere sufficienti buone API, mentre gli ecosistemi IoT ricchi di funzionalità e in rapida evoluzione di solito richiedono agli sviluppatori un controllo maggiore dell'intero sistema, del suo codice sorgente, delle interfacce di integrazione, delle opzioni di distribuzione, schemi di dati, connettività e meccanismi di sicurezza, ecc.;
- Sicuro: la sicurezza dei dati implica crittografia, gestione completa dell'identità e implementazione flessibile.
  Crittografia del flusso di dati *end-to-end*, inclusi i dati inattivi, l'autenticazione dei dispositivi, la gestione dei diritti di accesso degli utenti e l'infrastruttura cloud privata per i dati sensibili: queste sono le basi per evitare potenziali violazioni della soluzione IoT.

Attraversando questi aspetti, esistono due diversi paradigmi di implementazione del cluster di soluzioni IoT offerti dai provider di piattaforme IoT:

- un cloud pubblico IoT PaaS[5];
- un cloud IoT privato ospitato.

3.1.4. Cosa può fare un'azienda con una piattaforma IoT



Una piattaforma IoT svolge un ruolo chiave per i venditori e le startup di dispositivi "*smart*", che possono utilizzarla per dotare i propri prodotti di funzioni di controllo remoto e monitoraggio in tempo reale, avvisi e notifiche configurabili, servizi cloud collegabili e integrazione con gli smartphone e altri dispositivi dei consumatori.

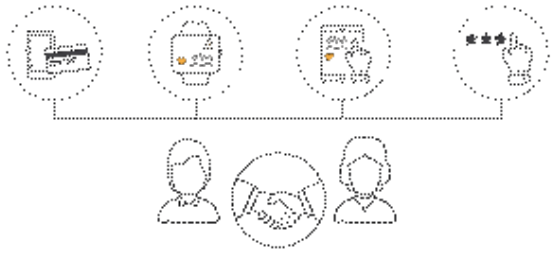
Un'altra applicazione della piattaforma IoT è l'ottimizzazione dei costi per le aziende nei settori: industriale, agricolo e dei trasporti; attraverso il monitoraggio remoto di dispositivi e veicoli, la manutenzione predittiva delle apparecchiature, la raccolta di dati dei sensori per l'analisi della produzione in tempo reale e la garanzia di sicurezza e tracciamento consegna merci "*end-to-end*".

I cloud IoT su larga scala sono soluzioni tipiche per CSP[6], *smart city* ecc.
Utilizzando una piattaforma IoT, queste aziende sviluppano infrastrutture IoT per fornire tutti i tipi di nuovi servizi per clienti regolari, società di servizi pubblici e società giganti.

[5]**PaaS**: Platform as a service (PaaS) è un modello di cloud computing in cui un provider di terze parti fornisce strumenti hardware e software (in genere quelli necessari per lo sviluppo di applicazioni) agli utenti su Internet.
Un provider PaaS ospita l'hardware e il software sulla propria infrastruttura.
Di conseguenza, PaaS libera gli sviluppatori dalla necessità di installare hardware e software interni per sviluppare o eseguire una nuova applicazione.

[6]**Content Security Policy (CSP)**: Standard di sicurezza del computer introdotto per prevenire script "*cross-site*" (XSS), "*clickjacking*" e altri attacchi di iniezione di codice risultanti dall'esecuzione di contenuti dannosi nel contesto di una pagina Web affidabile.
CSP fornisce un metodo standard per i proprietari di siti Web per dichiarare le origini approvate del contenuto che i browser dovrebbero essere autorizzati a caricare su quel sito Web.

Tra questi ci sono servizi di auto connesse, misurazione della rete intelligente, monitoraggio della qualità dell'aria in tutta la città, installazioni di edifici intelligenti e molti altri.



Infine, una piattaforma IoT è la tecnologia essenziale per migliorare l'esperienza del cliente nei settori della vendita al dettaglio, dell'assistenza sanitaria, dell'ospitalità e dei viaggi.

Viene utilizzato per consentire servizi altamente personalizzati e garantire l'interazione senza stress tra il cliente e l'azienda.

Un esempio è rappresentato dalle soluzioni di monitoraggio e trattamento a distanza dei pazienti, che sono incredibilmente convenienti da usare e consentono di risparmiare molto tempo a una persona durante le visite regolari in ospedale.

La raccolta di dati esaurienti sui pazienti diventa semplice con l'IoT, mentre i rivenditori e le aziende di ospitalità utilizzano una ricca raccolta di dati per creare offerte personali ed eseguire un marketing efficace.

3.2. Alcune piattaforme disponibili

Questo progetto è iniziato con una ricerca approfondita sulle varie piattaforme disponibili sul mercato per la monitorizzazione e l'immagazzinamento dei dati, dunque seguirà un elenco con le varie piattaforme ispezionate con una breve descrizione per ognuna di esse.

3.2.1. KAA IoT

Kaa è una tecnologia middleware IoT applicabile a qualsiasi IoT aziendale.

Fornisce una gamma di funzionalità che consentono agli sviluppatori di creare applicazioni avanzate per prodotti intelligenti, gestire in modo flessibile i loro ecosistemi (di dispositivi), orchestrare l'elaborazione dei dati *end-to-end* e molto altro.

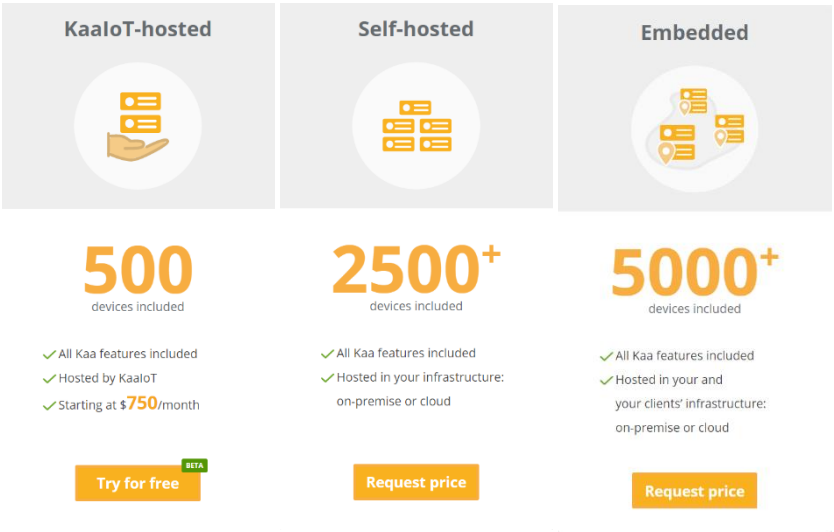


Figura 4.1: Piani proposti dalla piattaforma Kaa [1]

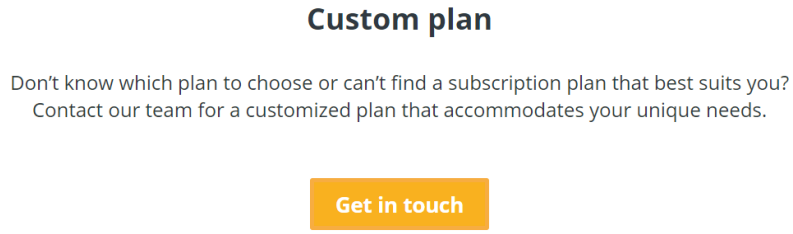


Figura 4.2: Piani proposti dalla piattaforma Kaa [2]

Subscription plans	INCLUDES	FEES
KAAIOT-HOSTED	500 Active Endpoints	$750/month
	100 GB storage	$0.60 per additional Active Endpoint
	5 Priority Engineering support requests per month	$100 per each additional Priority Engineering support requests

Figura 4.3: Tabella dei prezzi

3.2.1.1. Connettività

La connettività si basa sulla messaggistica tra il cloud e i dispositivi.
Ecco come i dispositivi si connettono al cloud per eseguire diverse operazioni.

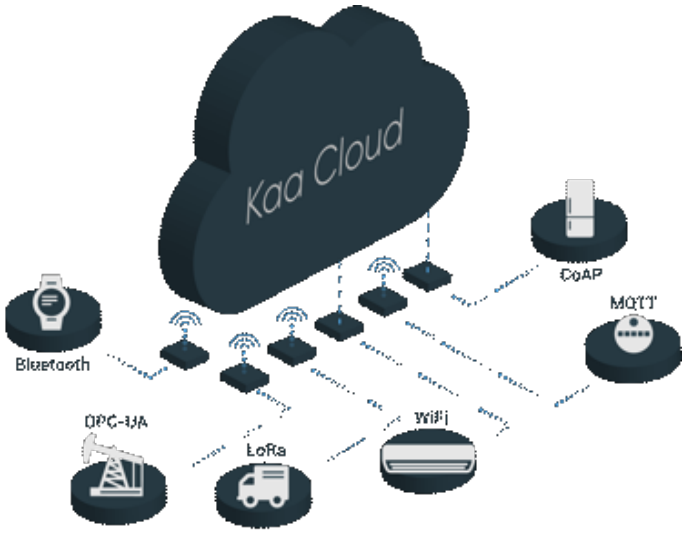


Figura 4.4: Esempio di connettività di Kaa IoT

La piattaforma Kaa supporta protocolli IoT leggeri per la connessione dei dispositivi, come MQTT e CoAP.
La piattaforma consente di creare applicazioni che funzionano su qualsiasi tipo di connessione di rete, sia persistente che intermittente.

Si può scegliere una delle implementazioni del protocollo di trasporto esistenti fornite con Kaa oppure creare trasporti personalizzati e collegarli al proprio sistema.

MQTT è il protocollo predefinito utilizzato da Kaa.
Il protocollo Kaa si basa su MQTT e CoAP e definisce le regole di base della comunicazione tra la piattaforma e i dispositivi. Il protocollo è completamente aperto, asincrono e consente formati di messaggi arbitrari.
Inoltre, si può scegliere tra canali crittografati, per proteggere i dati sensibili, e non crittografati, per i dati aperti.

Nel caso in cui il dispositivo non disponga di una connettività IP o implementi già alcune funzionalità di comunicazione, Kaa utilizza un'architettura gateway, in cui un gateway comunica con il dispositivo tramite un protocollo di rete locale o un protocollo di prossimità ed esegue la conversione dei messaggi a livello di trasporto o addirittura li rappresenta nel cloud.

3.2.1.2. Gestione dei dispositivi

Kaa fornisce un registro di gemelli digitali, che rappresentano cose, dispositivi e altre entità gestite dalla piattaforma.

Kaa consente di memorizzare gli attributi del dispositivo, che forniscono informazioni più dettagliate su qualsiasi caratteristica del dispositivo (Es.: Il numero di serie, l'indirizzo MAC, l'ubicazione, la versione del software, ecc.).
Oltre ai tipi di dati semplici, gli attributi possono contenere oggetti più complessi e strutturati, come un elenco di periferiche connesse e le loro proprietà.

Inoltre, è possibile costruire filtri basati sugli attributi del dispositivo per segmentare la popolazione del proprio dispositivo in gruppi gestiti individualmente.
In Kaa, i filtri vengono automaticamente rivalutati ogni volta che cambiano gli attributi del dispositivo.

Per connettersi alla piattaforma, un dispositivo deve presentare credenziali valide (Es.: chiavi precondivise, token, combinazioni di login e password, certificati, ecc.).
È possibile utilizzare le API di gestione delle credenziali Kaa per eseguire il provisioning, sospendere o revocare l'accesso.

Kaa monitora il dispositivo durante tutto il suo ciclo di vita, dagli eventi iniziali di provisioning e connettività agli aggiornamenti software e alla disattivazione finale.
È possibile creare e assegnare gestori personalizzati per automatizzare i flussi di lavoro corrispondenti.

3.2.1.3. Raccolta dati

Immediatamente, Kaa fornisce un protocollo facile da usare per la raccolta di dati dai dispositivi collegati.
Questo protocollo garantisce la consegna affidabile dei dati con codici di risposta, che indicano il risultato dell'elaborazione dei dati da parte della piattaforma.

Una volta ricevuti dalla piattaforma, i dati del dispositivo possono essere inviati a più *pipeline* di elaborazione.
Nel caso in cui si verifichino errori durante l'elaborazione, l'arresto del disco o il sovraccarico del processore, il dispositivo viene informato di

ciò. Di conseguenza, il dispositivo sa sempre se i dati inviati sono sicuri da eliminare o devono essere rinviati.

Per ridurre al minimo l'utilizzo della rete e migliorare la velocità di trasmissione dei dati, il protocollo supporta il batch[7]. Fornisce ai tuoi dispositivi la capacità di bufferizzare i dati localmente prima di caricarli in un messaggio.

I gateway intermedi possono eseguire la funzione di memorizzazione e inoltro. Oltre a ottimizzare l'efficienza della rete, questa funzionalità è utile nelle distribuzioni IoT con connettività intermittente e aiuta a preservare la durata della batteria del dispositivo.

Kaa consente di raccogliere dati strutturati e non strutturati. Possono essere di tipo primitivi, come numeri semplici o testo, o composti, come mappe dei valori-chiave, matrici o oggetti nidificati.

3.2.1.4. Elaborazione, analisi e visualizzazione dei dati

Kaa offre molta libertà nel trattamento dei dati raccolti.
La piattaforma presenta adattatori di raccolta dati che consentono l'invio di dati a vari database o sistemi di analisi dei dati.

Grazie ad un'architettura altamente modulare di Kaa, una nuova integrazione richiede il minimo.

I dati grezzi e non strutturati possono anche essere trasformati in serie temporali ben strutturate, utili per: analisi, analisi dei modelli, visualizzazione, creazione di grafici, ecc.

Analogamente ai dati grezzi, le serie temporali possono essere inserite in un sistema di elaborazione o analisi di propria scelta.
Oltre a visualizzare il valore principale per una serie temporale (Es.: la temperatura), la piattaforma consente agli utenti di impostare tag, che consentono la visualizzazione di alcuni dati aggiuntivi, come posizione, intensità della luce, umidità, ecc.

I valori dei tag vengono estratti dai dati grezzi raccolti e allegati a ciascun punto dati in una serie temporale ed è possibile creare più serie temporali da un campione di dati.

Il componente di visualizzazione dei dati di Kaa comprende un ricco set di widget, come indicatori, grafici, mappe, tabelle, ecc.

[7]**Batch**: In informatica, il termine batch viene utilizzato con significati specifici, tipicamente riferite a uno o più dei seguenti aspetti del batch processing:

- la non interattività dei programmi;
- l'esecuzione "accorpata" di più programmi;
- l'esecuzione non immediata, ma rimandata nel tempo dei programmi;
- la modifica di più dati nello stesso momento.

L'uso più diffuso del termine è probabilmente quello riferito a un insieme di comandi o programmi, tipicamente non interattivi, aggregati per l'esecuzione, come in uno script o un comando batch.

È possibile utilizzare questi widget per visualizzare diversi tipi di dati, sia telemetria, statistica, geolocalizzazione, metadati, filtri, aggiornamenti software o altro, sia storico che attuale.

Tutti i widget sono configurabili e consentono di modificare le loro fonti di dati e la rappresentazione visiva.

Oltre alla visualizzazione dei dati, i widget consentono di interagire con i dispositivi inviando comandi, modificando configurazione e metadati, ecc.

Le dashboard aiutano ad organizzare i widget in gruppi logici e a definirne il layout.
Queste possono essere collegate ipertestualmente per semplificare la navigazione nei complessi set di dati multi-dispositivo. Inoltre, Kaa supporta il modello di dashboard, che consente di riutilizzare una configurazione per più dashboard del dispositivo.

Grazie alle sue API aperte per l'integrazione di sistemi di terze parti, Kaa può essere collegato a strumenti di visualizzazione ed esplorazione dei dati di propria scelta.

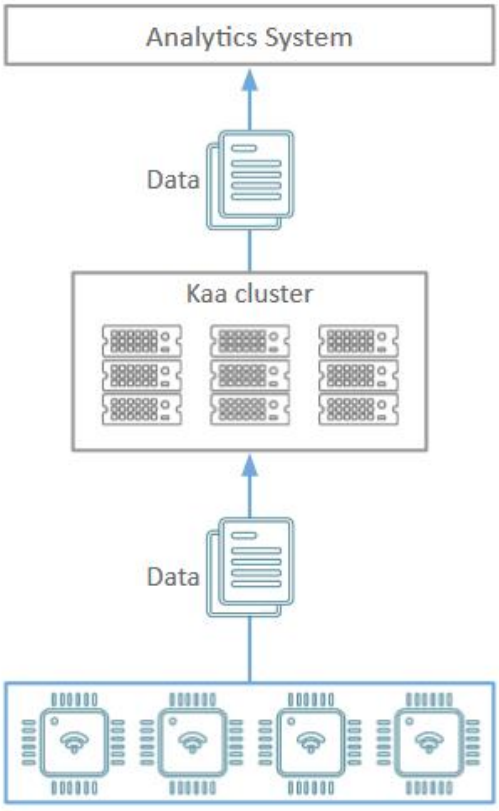


Figura 4.5: Schematizzazione della raccolta dati di Kaa IoT

3.2.2. Oracle Integration Cloud

Oracle Integration è una piattaforma unificata in cui è possibile integrare le proprie applicazioni, automatizzare i processi, creare applicazioni e analizzare i risultati.

Product	Pay As You Go (5K Messages/Hour)	Monthly Flex (5K Messages/Hour)	Part Number	Features included
Oracle Integration Cloud Service - Standard	€1.0864	€0.7243	B89639	• Integration ◦ SaaS Integration Adapters ◦ Technology Adapters ◦ Scheduled File Transfer • Visual Builder ◦ Business Object Modeler
Oracle Integration Cloud Service - Enterprise	€2.1729	€1.4486	B89640	• Integration ◦ Same as Standard + On premise Application Adapters • Process ◦ Process Automation • Visual Builder ◦ Same as Standard

Figura 4.6: Tabella dei prezzi

3.2.2.1. Panoramica

In particolare, con Oracle Integration, si può:

- Utilizzare integrazioni per progettare, monitorare e gestire le connessioni tra le proprie applicazioni;
- Creare applicazioni di processo per automatizzare e gestire i flussi di lavoro aziendali;
- Creare applicazioni Web e mobili personalizzate;
- Analizzare i risultati per ottenere informazioni dettagliate sulla propria attività.

I processi aziendali critici, come quelli relativi alla gestione del capitale umano (HCM), esperienza del cliente (CX) e pianificazione delle risorse aziendali (ERP), sono spesso lenti e poco flessibili.

Ad esempio, un processo in più fasi come "*Lead to Opportunity to Quote to Order*" può coinvolgere quattro o più applicazioni e richiedere la gestione delle eccezioni umane in ogni fase del processo.
In questo scenario, la mancanza di integrazione tra i reparti e i ritardi causati dalla risoluzione dei problemi basata sull'uomo possono comportare perdite di entrate, clienti frustrati e costi elevati.

Oracle Integration consente di:

- Stabilire la connettività tra le molte applicazioni e le persone che fanno parte dell'intero ciclo di vita dei processi aziendali.
- Assemblare le tecnologie esistenti in nuovi servizi aziendali per allinearsi meglio con il ritmo mutevole delle nuove esigenze aziendali.

- Offrire nuove innovazioni aziendali più rapidamente collegando rapidamente diverse applicazioni e ruoli aziendali chiave.

3.2.2.2. "*Integration*": collegamento di applicazioni

È possibile utilizzare le integrazioni per connettere le applicazioni in un flusso aziendale continuo.

È possibile sviluppare e attivare rapidamente integrazioni tra le applicazioni che vivono nel cloud e le applicazioni che vivono ancora nei locali.

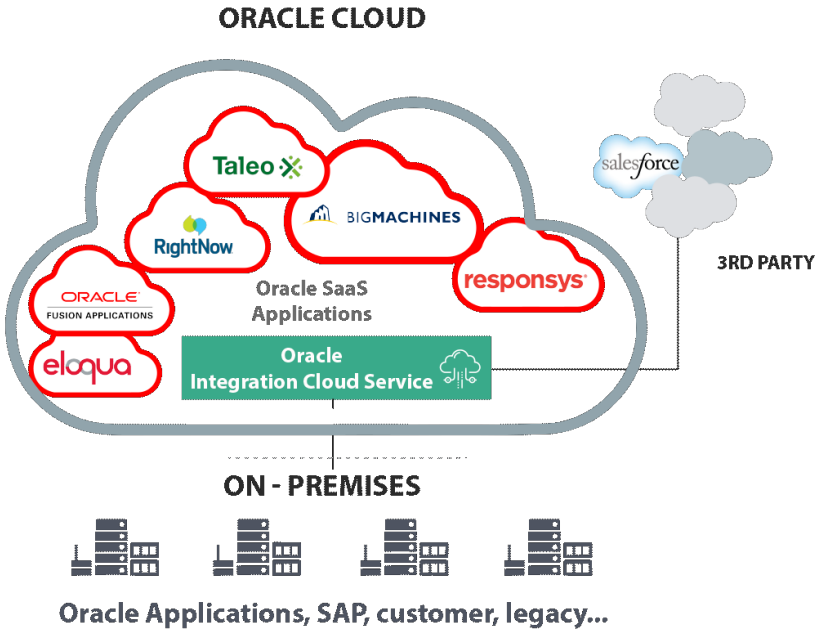


Figura 4.7: Oracle Integration Cloud

- Identificare le applicazioni che si desidera integrare e configurare i dettagli di connessione per ciascuna applicazione.
- Utilizzare strumenti visivi basati su browser per creare integrazioni e quindi mappare i dati tra le applicazioni.
  Le mappature possono variare da semplici assegnazioni di dati a espressioni complesse.
- Monitorare la dashboard per verificare lo stato e l'elaborazione delle statistiche per un'integrazione.
  La dashboard, inoltre, misura e tiene traccia delle prestazioni delle proprie transazioni acquisendo e segnalando le informazioni chiave.

3.2.3. Amazon Web Services (AWS) IoT Core

AWS IoT Core è un servizio cloud gestito che consente ai dispositivi connessi di interagire facilmente e in modo sicuro con le applicazioni cloud e altri dispositivi.

AWS IoT Core può supportare miliardi di dispositivi, trilioni di messaggi e può elaborare e instradare tali messaggi agli endpoint AWS e ad altri dispositivi in modo affidabile e sicuro.

Con AWS IoT Core, le applicazioni possono tenere traccia e comunicare con tutti i dispositivi anche quando questi non sono connessi.

AWS IoT Core semplifica inoltre l'utilizzo dei servizi AWS (Es.: AWS Lambda, Amazon Kinesis, Amazon S3, Amazon SageMaker ecc.) per creare applicazioni IoT che raccolgono, elaborano, analizzano dati generati da dispositivi connessi, senza la necessità della gestione di nessuna infrastruttura.

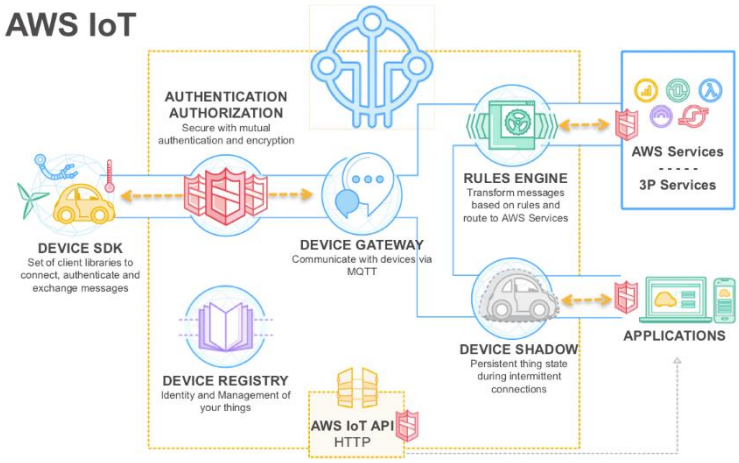


Figura 4.8: Schema di AWS IoT

AWS offre una prova gratuita.
Offre anche un approccio "*pay-as-you-go*" per il prezzo, il che consiste nel pagare mentre si utilizza e solo per ciò che si utilizza e per oltre 160 servizi di cloud.
Con AWS si paga unicamente per i servizi necessari e solo per il tempo di utilizzo senza richiesta di un contratto a lungo temine o licenze complesse.
Il pagamento viene infatti paragonato ai servizi come acqua ed elettricità.

3.2.3.1. Come funziona AWS IoT Core

Connettere e gestire i propri dispositivi

AWS IoT Core consente di connettere facilmente i dispositivi al cloud e ad altri dispositivi.

AWS IoT Core supporta HTTP, WebSocket[8] e MQTT, protocolli di comunicazione leggeri appositamente progettato per tollerare connessioni intermittenti, ridurre al minimo l'impronta del codice sui dispositivi e ridurre i requisiti di larghezza di banda della rete.

AWS IoT Core supporta anche altri protocolli standard e personalizzati del settore e i dispositivi possono comunicare tra loro anche se utilizzano protocolli diversi.

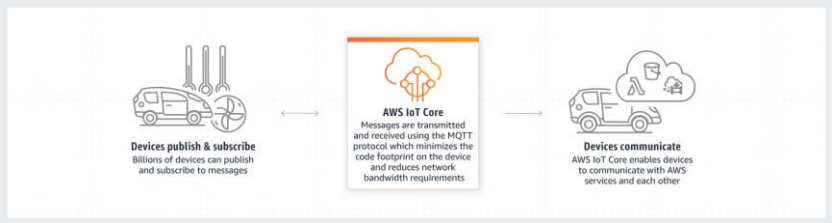


Figura 4.9: Schematizzazione del funzionamento di AWS IoT Core [1]

Connessioni e dati del dispositivo sicuri

AWS IoT Core fornisce autenticazione e crittografia *end-to-end* in tutti i punti di connessione, in modo che i dati non vengano mai scambiati tra dispositivi e AWS IoT Core senza una comprovata identità.

Inoltre, è possibile proteggere l'accesso ai propri dispositivi e applicazioni applicando criteri con autorizzazioni granulari.

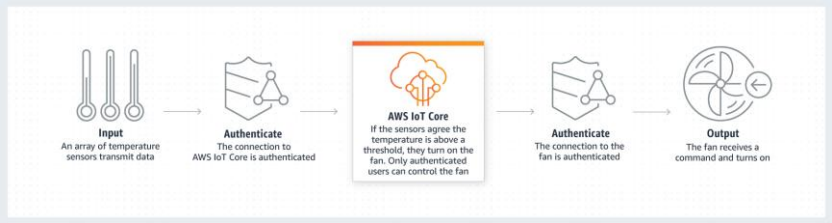


Figura 4.10: Schematizzazione del funzionamento di AWS IoT Core [2]

Elaborare e agire sui dati del dispositivo

Con AWS IoT Core è possibile filtrare, trasformare e agire al volo sui dati dei dispositivi, in base alle regole aziendali che definisci.

Si possono aggiornare le regole per implementare nuove funzionalità di dispositivi e applicazioni in qualsiasi momento.

[8] **WebSocket**: Protocollo di comunicazione per computer, che fornisce canali di comunicazione full duplex su una singola connessione TCP.
Il protocollo WebSocket è stato standardizzato dall'IETF come **RFC 6455** nel 2011 e l'API WebSocket in Web IDL è stata standardizzata dal W3C.

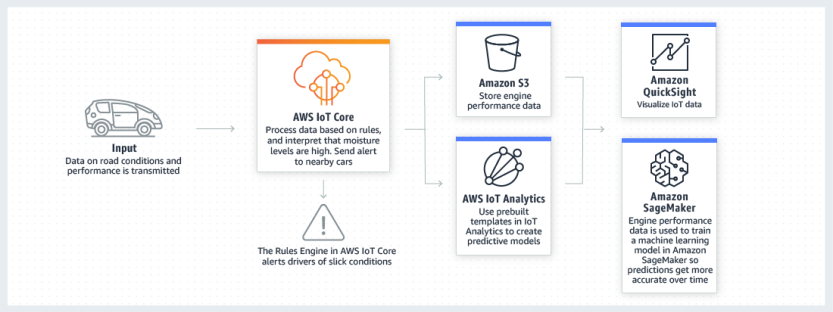


Figura 4.11: Schematizzazione del funzionamento di AWS IoT Core [3]

Leggere e impostare lo stato del dispositivo in qualsiasi momento

AWS IoT Core memorizza lo stato più recente di un dispositivo connesso in modo che possa essere letto o impostato in qualsiasi momento, facendo apparire il dispositivo alle applicazioni come se fosse sempre online.

Ciò significa che l'applicazione può leggere lo stato di un dispositivo anche quando è disconnesso e consente di impostare uno stato del dispositivo e di averlo implementato quando il dispositivo si riconnette.

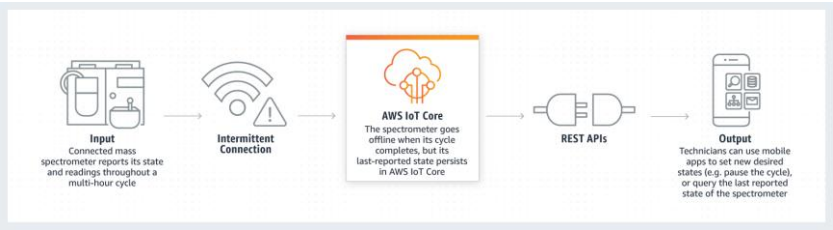


Figura 4.12: Schematizzazione del funzionamento di AWS IoT Core [4]

3.2.4. Google Cloud IoT

Google Cloud IoT è un set completo di strumenti per connettere, elaborare, archiviare e analizzare i dati sia ai margini che nel cloud.

La piattaforma è composta da servizi cloud scalabili e completamente gestiti; uno *stack* software integrato per l'elaborazione "*edge/on-premise*" con funzionalità di apprendimento automatico per tutte le esigenze dell'IoT.

È un servizio a pagamento ed offre la possibilità di una prova gratuita del valore di $300 per 12 mesi. I prezzi di Cloud IoT Core vengono calcolati in base al volume di dati utilizzato in un mese di calendario:

Volume di dati mensile	Prezzo per MB	Numero dispositivi registrati	Tariffa minima*
Fino a 250 MB	$ 0,00	Illimitato, entro i limiti massimi di QPS	1024 byte
Da 250 MB a 250 GB	$ 0,0045	Illimitato, entro i limiti massimi di QPS	1024 byte
Da 250 GB a 5 TB	$ 0,0020	Illimitato, entro i limiti massimi di QPS	1024 byte
Oltre 5 TB	$ 0,00045	Illimitato, entro i limiti massimi di QPS	1024 byte

Figura 4.13: Tabella costi GC IoT

3.2.4.1. Panoramica

Google Cloud IoT Core, in combinazione con altri servizi sulla piattaforma Cloud IoT, fornisce una soluzione completa per la raccolta, l'elaborazione, l'analisi e la visualizzazione dei dati IoT in tempo reale per supportare una migliore efficienza operativa.

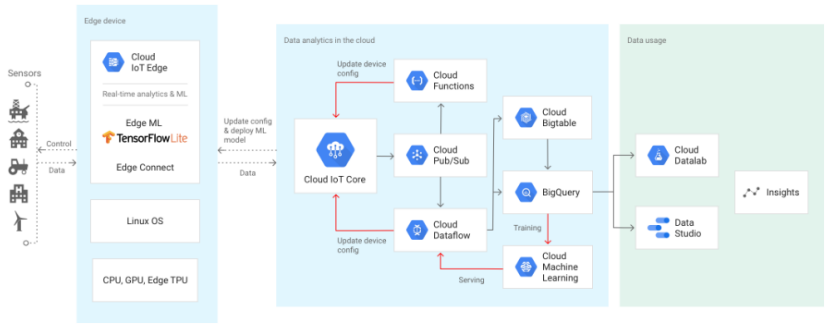


Figura 4.14: Schema GC IoT Core

Cloud IoT Core, utilizzando Cloud "*Pub/Sub*", può aggregare i dati del dispositivo dispersi in un unico sistema globale che si integra perfettamente con i servizi di analisi dei dati di Google Cloud.

È possibile utilizzare il proprio flusso di dati IoT per analisi avanzate, visualizzazioni, apprendimento automatico e altro ancora per migliorare l'efficienza operativa, anticipare i problemi e creare modelli ricchi che descrivono e ottimizzano meglio la tua attività.

Cloud IoT Core supporta i protocolli MQTT e HTTP standard, in modo da poter utilizzare i dispositivi esistenti con modifiche minime al firmware.

Cloud IoT Core funziona sull'infrastruttura serverless di Google, che si ridimensiona automaticamente in risposta alle modifiche in tempo reale e aderisce ai rigorosi protocolli di sicurezza standard del settore che proteggono i dati aziendali.

3.2.4.2. Funzionalità incluse

- **Gestore dispositivi:** Consente ai singoli dispositivi di essere configurati e gestiti in modo sicuro; la gestione può essere effettuata tramite una console o programmaticamente.
  La gestione dei dispositivi stabilisce l'identità di un dispositivo e fornisce il meccanismo per autenticare un dispositivo durante la connessione.
  Mantiene inoltre una configurazione logica di ciascun dispositivo e può essere utilizzato per controllare in remoto il dispositivo dal cloud.
- **Protocollo bridge:** Fornisce endpoint di connessione per protocolli con bilanciamento del carico automatico per tutte le connessioni del dispositivo.
  Il bridge di protocollo supporta nativamente la connessione sicura su protocolli standard del settore come MQTT e HTTP.
  Il bridge di protocollo pubblica tutti i dati di telemetria dei dispositivi su Cloud "*Pub/Sub*", che possono quindi essere utilizzati dai sistemi analitici a valle.
- **Sicurezza "*end-to-end*":** I dispositivi che supportano i requisiti di sicurezza del cloud IoT Core possono offrire una sicurezza "*full-stack*".
- **Sistema globale unico:** Collega tutti i dispositivi e i gateway a Google Cloud tramite protocolli standard, come MQTT e HTTP, attraverso gli endpoint del protocollo e gestisci tutti i dispositivi come un unico sistema globale.
  Il servizio utilizza Cloud Pub/Sub, che conserva i dati per sette giorni.
  È anche possibile ridimensionare istantaneamente senza limiti, utilizzando il ridimensionamento orizzontale della piattaforma Google Cloud.
- **Completamente gestito e scalabile:** Il servizio è senza server e non richiede alcuna installazione iniziale del software.
- **Distribuzione del dispositivo su larga scala:** È possibile utilizzare le API REST per gestire automaticamente la registrazione, la distribuzione e il funzionamento dei dispositivi su vasta scala.
  Inoltre, si possono utilizzare le API per recuperare e aggiornare le proprietà e lo stato del dispositivo anche quando i dispositivi non sono connessi.

- **Abilita comunicazione ad alta frequenza e bassa latenza:** Invia direttive di comando o configurazione ai dispositivi collegati a Cloud IoT Core.
  I comandi sono direttive veloci, frequenti e una tantum inviate ai dispositivi.
  Le configurazioni sono direttive persistenti che, quando si utilizza MQTT, vengono inviate a tutti i dispositivi sottoscritti, anche quelli aggiunti in seguito.
- **Funzionamento offline e supporto per dispositivi con risorse limitate:** È possibile utilizzare un gateway per fornire funzionalità operative offline ai dispositivi con risorse limitate.
  Un gateway può eseguire attività per conto di un dispositivo, inclusa la comunicazione con Cloud IoT Core, la connessione a Internet e l'autenticazione delle credenziali.
- **Metriche in tempo reale con Stackdriver Monitoring:** Utilizza il monitoraggio Stackdriver per creare dashboard che mostrano dati come il numero totale di dispositivi attivi in un registro.
  Inoltre, imposta avvisi basati su soglie metriche, ad esempio i dispositivi in un registro che superano un limite di dati fatturabili predefinito.
- **Tutto il tuo dispositivo registra in un unico posto:** Controlla i log di connessione ed errori in Stackdriver Logging accanto ai log di controllo.
  Configura metriche definite dall'utente per ottenere approfondimenti come il numero di dispositivi che hanno pubblicato i dati su uno specifico argomento Cloud Pub/Sub.

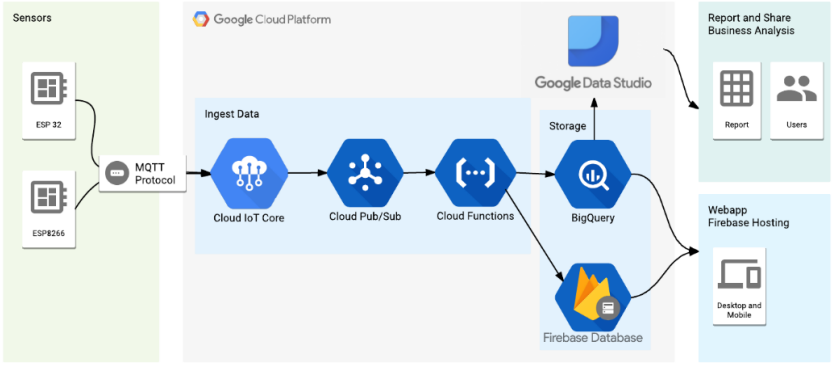


Figura 4.15: Schema funzionalità GC IoT

[9]**Software as a service (SaaS)**: Modello di distribuzione del software applicativo dove un produttore di software sviluppa, opera (direttamente o tramite terze parti) e gestisce un'applicazione web che mette a disposizione dei propri clienti via Internet (previo abbonamento); spesso si tratta di un servizio di cloud computing.

3.2.5. Azure IoT

Azure Internet of Things (IoT) è una raccolta di servizi cloud gestiti da Microsoft che connettono, monitorano e controllano miliardi di asset IoT.

In termini più semplici, una soluzione IoT è costituita da uno o più dispositivi IoT e uno o più servizi *back-end* in esecuzione nel cloud che comunicano tra loro.

Per Device Pricing (Each Device comes with 50,000 included messages per month)	0 - 5 6 - 1,000 1,001 - 10,000 10,001 - 100,000 100,001+	Free €1.7 per device/month €1.3 per device/month €0.9 per device/month €0.5 per device/month
Additional Message Pricing[1]	per 1M messages	€4.217

Figura 4.16: Tabella Prezzi Azure IoT

3.2.5.1. Tecnologie e soluzioni per l'Internet of Things (IoT): PaaS e SaaS

Microsoft ha creato un portafoglio che supporta le esigenze di tutti i client, consentendo a tutti di accedere ai vantaggi della trasformazione digitale.

Il portafoglio di prodotti IoT di Azure offre una panoramica delle tecnologie e delle soluzioni PaaS/SaaS[9] disponibili. Presenta i due percorsi disponibili per creare la soluzione:

- **Platform as a Service (PaaS):** è possibile creare la propria applicazione utilizzando uno dei seguenti servizi:
  - Acceleratori di soluzioni IoT di Azure, che sono una raccolta di soluzioni preconfigurate di livello aziendale che consentono di accelerare lo sviluppo di soluzioni IoT personalizzate o
  - Il servizio Azure Digital Twins, che consente di modellare l'ambiente fisico per creare soluzioni IoT sensibili al contesto usando un grafico di intelligenza spaziale e modelli di oggetti specifici del dominio.
- **Software as a Service (SaaS):** si può iniziare velocemente con Azure IoT Central, la nuova soluzione SaaS per sviluppare applicazioni IoT senza essere esposto alla complessità della soluzione IoT.

Se l'organizzazione non dispone delle risorse per creare la propria soluzione IoT, Azure IoT Central è una soluzione IoT senza codice che può creare modelli di dispositivo, dashboard e regole in pochi minuti.

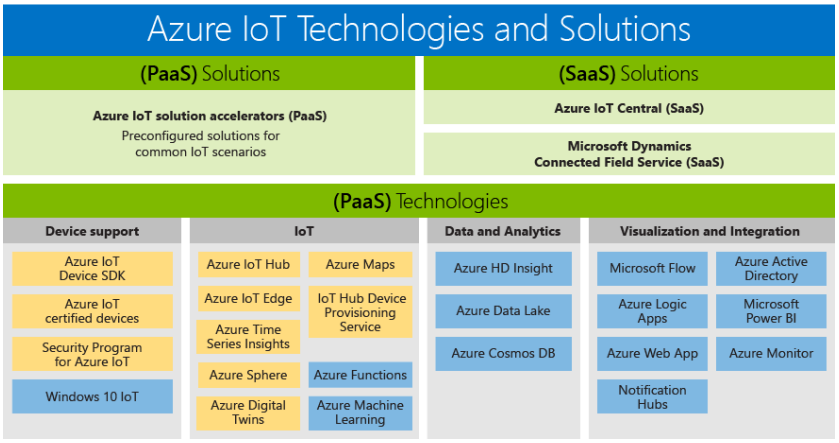


Figura 4.17: Schematizzazione servizi Azure IoT

3.2.5.2. Soluzioni

È possibile iniziare rapidamente con acceleratori di soluzioni e offerte SaaS, scegliendo tra soluzioni preconfigurate che consentono scenari IoT comuni, come monitoraggio remoto e manutenzione predittiva per creare una soluzione completamente personalizzabile.
Oppure è possibile utilizzare Azure IoT Central, una soluzione *end-to-end* completamente gestita che abilita potenti scenari IoT senza richiedere competenze in soluzioni cloud.

Azure IoT solution accelerators (PaaS)

Gli acceleratori di soluzione IoT di Azure sono soluzioni PaaS personalizzabili che offrono un alto livello di controllo sulla soluzione IoT.

Le organizzazioni con un gran numero di dispositivi o modelli di dispositivi e i produttori che cercano soluzioni di fabbrica connesse sono esempi di aziende che possono trarre vantaggio dagli acceleratori di soluzioni IoT.

Creando soluzioni altamente personalizzabili su misura per esigenze complesse, gli acceleratori di soluzioni IoT forniscono:

- Soluzioni precompilate;
- Monitoraggio remoto;
- Fabbrica collegata;

[10]SDK: Raccolta di strumenti di sviluppo software in un pacchetto installabile.
Facilitano la creazione di applicazioni con compilatore, debugger e forse un framework software.
Normalmente sono specifici per una combinazione di piattaforma hardware e sistema operativo.

- Manutenzione predittiva;
- Simulazione del dispositivo;
- Possibilità di distribuire in pochi minuti;
- Tempo di valutazione accelerato;
- Soluzioni che offrono il massimo controllo.

Azure IoT Central (SaaS)

Azure IoT Central è una soluzione SaaS completamente gestita, che consente di iniziare rapidamente con un'esperienza IoT minima.

Le organizzazioni con un numero ristretto di dispositivi, scenari più prevedibili e capacità IoT/IT limitate possono sfruttare i vantaggi dell'IoT attraverso un approccio SaaS.
Le aziende che in precedenza mancavano di tempo, denaro e competenza per sviluppare prodotti connessi, ora possono iniziare rapidamente con Azure IoT Central.

Microsoft è leader del settore nel fornire una soluzione SaaS matura che soddisfi i requisiti di implementazione IoT comuni.

- SaaS IoT completamente gestito;
- Non è richiesta alcuna competenza nello sviluppo di soluzioni cloud;
- Configurabile per le tue esigenze;
- Ideale per le semplici esigenze dell'IoT.

3.2.5.3. Tecnologie (PaaS)

Con il portafoglio IoT più completo di servizi di piattaforma, le tecnologie Platform-as-a-Service (PaaS) che abbracciano la piattaforma Azure consentono di creare, personalizzare e controllare facilmente tutti gli aspetti della soluzione IoT.

È possibile stabilire comunicazioni bidirezionali con miliardi di dispositivi IoT e gestire i dispositivi IoT su larga scala. Quindi integrare i dati del dispositivo IoT con altri servizi della piattaforma, come Azure Cosmos DB e Azure Time Series Insights, per migliorare le informazioni sulla soluzione.

Supporto dispositivo

Tutti i dispositivi sono indipendenti dalla piattaforma e testati per connettersi perfettamente all'hub IoT.
Si possono connettere tutti i dispositivi ad Azure IoT usando gli SDK[10] per dispositivi open source. Gli SDK supportano più sistemi operativi, come Linux, Windows e sistemi operativi in tempo reale, oltre a più linguaggi di programmazione come C, Node.js, Java, .NET e Python.

Hub IoT

L'hub IoT di Azure è un servizio completamente gestito che consente comunicazioni bidirezionali affidabili e sicure tra milioni di dispositivi IoT e un *back-end* della soluzione.

Il servizio di provisioning dei dispositivi dell'hub IoT di Azure è un servizio di supporto per l'hub IoT che consente il provisioning zero-touch[11] e just-in-time[12] sull'hub IoT corretto senza richiedere l'intervento umano, consentendo ai clienti di fornire milioni di dispositivi in modo sicuro e scalabile.

Edge

Azure IoT Edge è un servizio IoT. Questo servizio è destinato ai clienti che desiderano analizzare i dati sui dispositivi, ad esempio "al limite".

Spostando parti del carico di lavoro al limite, si verificherà una latenza ridotta e si avrà la possibilità di scenari off-line.

Intelligenza spaziale

Azure Digital Twins è un servizio IoT che consente di creare un modello di ambiente fisico. Fornisce un grafico di intelligenza spaziale per modellare le relazioni tra persone, spazi e dispositivi.

Correlando i dati nel mondo fisico e digitale è possibile creare soluzioni contestualmente consapevoli.

Dati e analisi

Sfrutta una vasta gamma di offerte PaaS di dati e analisi di Azure nella tua soluzione IoT, dall'intelligenza del cloud al limite con Azure Machine Learning, alla memorizzazione dei dati dei dispositivi IoT in modo conveniente con Azure Data Lake, per visualizzare enormi quantità di dati dai dispositivi IoT con Azure Time Series Insights.

Visualizzazione e integrazione

Microsoft Azure offre una soluzione cloud completa, che combina una raccolta in costante crescita di servizi cloud integrati con un impegno leader del settore per la protezione e la privacy dei dati.

[11] **Zero-touch**: Funzione switch che consente di effettuare il provisioning e la configurazione automatica dei dispositivi, eliminando gran parte del lavoro manuale necessario per aggiungerli a una rete.

[12] **Just-in-time**: Fornisce un altro modo per eseguire il provisioning degli utenti nel servizio VMware Identity Manager. Invece di sincronizzare gli utenti da un'istanza di Active Directory, con il provisioning Just-in-Time gli utenti vengono creati e aggiornati dinamicamente quando effettuano l'accesso, in base alle asserzioni SAML inviate dal provider di identità.

3.2.6. Altair SmartCore

3.2.6.1. Panoramica

Altair SmartCore fa parte della suite Altair SmartWorks ™, una soluzione ad architettura aperta che consente applicazioni IoT "*Edge-to-Cloud*" avanzate e analisi dei dati potenziate basate sull'apprendimento automatico per guidare l'innovazione.

Figura 4.18: Esempio di creazione con Altair SmartCore

Altair SmartCore è una piattaforma nativa cloud, che offre un set integrato di servizi e funzionalità per aiutare a connettere facilmente le proprie cose al mondo digitale.

Disponibile come piattaforma come servizio (PaaS) o locale, Altair SmartCore aiuterà a eseguire i propri progetti IoT più velocemente in un ambiente facile da usare, affidabile e altamente scalabile.

3.2.7. Piattaforma Ubidots

Ubidots è una società a due piattaforme: Ubidots e Ubidots STEM.
Il piano STEM standard è una licenza non commerciale che supporta, quello che viene definita dal team di Ubidots, la loro missione di rendere l'IoT più semplice e accessibile a tutti ed è utilizzato da migliaia di studenti, ricercatori e hobbisti di tutto il mondo.
Ubidots STEM si applica meglio a progetti fai da te e usi non commerciali.
I progetti rivolti ad applicazioni commerciali o industriali dovrebbero utilizzare gli strumenti di sviluppo avanzati e l'affidabilità della piattaforma Ubidots principale, anche nelle fasi di prototipazione. Per le applicazioni che richiedono più di tre dispositivi, la piattaforma Ubidots è probabilmente la soluzione migliore.

3.2.7.1. Ubidots

Le licenze Ubidots consentono agli integratori di sistemi, agli OEM[13] e agli imprenditori IoT di creare e distribuire applicazioni e soluzioni IoT in modo rapido e con il minimo sforzo di codifica.
Con Ubidots un amministratore può offrire portali IoT personalizzati per i clienti finali, sviluppare applicazioni IoT con maggiore affidabilità e supporto e monitorare un'intera distribuzione IoT da un singolo portale di amministrazione Ubidots.

Capacità

- Nessuna restrizione d'uso (può essere utilizzato per progetti commerciali o di ricerca);
- Da 1 a migliaia di dispositivi (varia per piano);
- Anni di stoccaggio e conservazione;
- SLA e piani di supporto disponibili;
- Meccanismi dedicati di inserimento dati, massimizzando i tempi di attività e i tempi di risposta delle API;
- Tempo di attività del 99,5% in tutti i sottosistemi (o più in uno SLA contrattato).

Funzionalità incluse

Tutte le funzionalità della licenza STEM, oltre a:

- **Gestione app:** crea più app, ognuna con dominio, logo e colori personalizzati;

[13]**OEM**: Abbreviazione di produttore di apparecchiature originali, termine utilizzato per descrivere una società che ha una relazione speciale con i produttori di computer e IT.
Gli OEM sono in genere produttori che rivendono i prodotti di un'altra azienda con il proprio nome e marchio.

[14]**Onboarding**: Noto anche come socializzazione organizzativa, è il gergo di gestione creato per la prima volta negli anni '70 che si riferisce al meccanismo attraverso il quale i nuovi dipendenti acquisiscono le conoscenze, le abilità e i comportamenti necessari per diventare membri organizzativi e addetti ai lavori efficaci.
Le metodologie utilizzate in questo processo includono incontri formali, lezioni, video, materiali stampati o orientamenti basati su computer per introdurre i nuovi arrivati ai loro nuovi posti di lavoro e organizzazioni.

- **IoT App Builder:** crea widget personalizzati usando il tuo HTML / JS e garantisce la possibilità di distribuirli su più dashboard;
- **Gestione dei dispositivi:** crea organizzazioni all'interno delle app, ognuna con i propri dispositivi, utenti, dashboard ed eventi;
- **Gestione utenti:** crea utenti e gestisce la propria app con diverse autorizzazioni e funzionalità di accesso in modo che gli utenti possano vedere ciò di cui hanno bisogno e niente di più;
- **Ubidots Analytics Tools:** esegue funzioni senza server o crea un API personalizzata per decodificare o codificare frame di dati usando UbiFunctions. Calcola la matematica e le espressioni statistiche direttamente dall'interfaccia utente di Ubidots con variabili sintetiche.
- Crea degli endpoint API personalizzati per accedere a dati, API e strumenti di terze parti utilizzando le richieste HTTPS;
- **Motore di eventi complessi:** attiva la logica di business e gli eventi dal motore di eventi condizionali complessi (SMS, Email, Telegram, Slack, Voice Calling) in Ubidots;
- È possibile utilizzare i tipi di dispositivo Ubidots per configurare proprietà, aspetto e variabili del dispositivo per automatizzare l'onboarding[14] di migliaia di dispositivi;
- Domini app privati;
- Personalizzazione del marchio e della visualizzazione di applicazioni private.

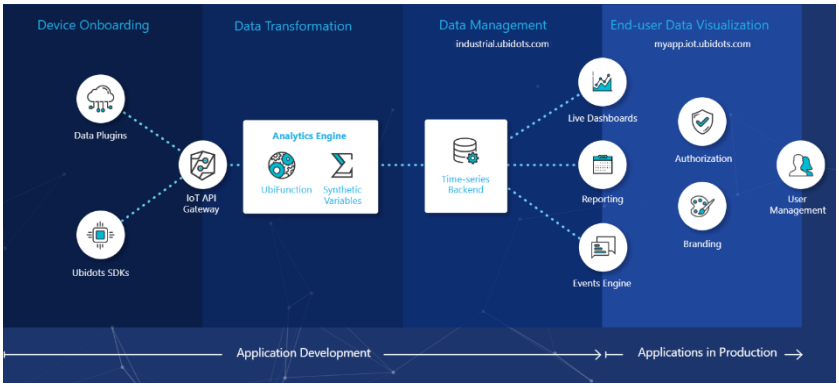


Figura 4.19: Infografica per comprendere meglio l'architettura di sviluppo e distribuzione di applicazioni IoT e gli strumenti di Ubidots

3.2.7.2. Ubidots STEM

Capacità

- Server condiviso e risorse computazionali tra tutti gli utenti STEM. Velocità e affidabilità basate sulle richieste totali della piattaforma in un dato secondo.
- SOLO per uso non commerciale (educazione personale, ricerca IoT o progetti fai da te)

- 1 dashboard (solo statico)
- Fino a 3 dispositivi, ogni dispositivo può avere fino a 10 variabili
- 1 mese di conservazione dei dati
- 4.000 punti al giorno
- Fino a 30 punti al minuto su tutti i dispositivi
- Supporto self-service con il Centro assistenza o la community di Ubidots.
- 10 SMS gratuiti e 1 chiamata vocale (USA e Canada) per dare il via ai propri progetti. È possibile aggiungere saldo al proprio account in qualsiasi momento per inviare più avvisi, inclusi SMS internazionali e chiamate vocali.

Funzionalità incluse

- Massimo 10 widget per dashboard
- REST API e supporto MQTT
- SMS, chiamate vocali, e-mail, avvisi WebHook, Slack & Telegram
- Calcolo di variabili sintetiche (funzioni speciali non incluse)

3.2.8. Horsa IoT Platform

3.2.8.1. Panoramica

Horsa è una importante realtà ICT italiana che progetta, implementa e governa soluzioni IT destinate alle imprese.

Horsa IoT Platform è un insieme di tecnologie, software e API che permettono di parlare con i propri oggetti fisici, qualsiasi essi siano. Una piattaforma pensata per le aziende, ad uso delle persone.

È possibile gestire gli oggetti intelligenti attraverso una Web Console oppure attraverso delle API.
Utilizza l'accentratore IOOOTA (Hub IoT) per l'integrazione degli oggetti fisici (quando necessario).
L'interazione con la piattaforma avviene attraverso tecnologie standard come HTTP, WebSocket, MQTT.
Memorizza i dati in un cluster Big Data (Hadoop, Spark, ecc.). Analizza in dati in real-time per la generazione di alert. Naviga grandi moli di dati e Analytics grazie a doolyk®.

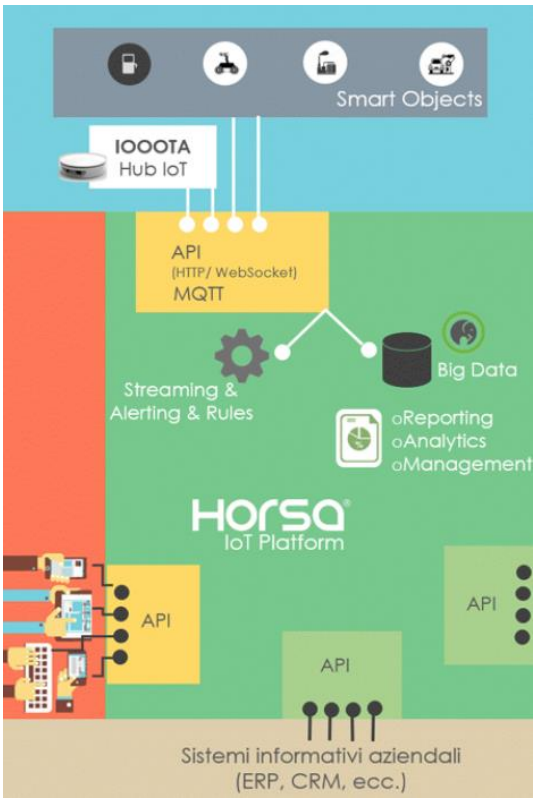


Figura 4.20: Schematizzazione di Horsa IoT

3.2.9. Piattaforma IoT universale HPE (HP Enterprise)

3.2.9.1. Panoramica

Con la piattaforma IoT universale di HPE è possibile ottenere una soluzione di settore verticale e indipendente dai client, scalabile, modulare e versatile. Questo consente di gestire le soluzioni IoT e offre valore attraverso la monetizzazione del grande quantitativo di dati generati dai dispositivi connessi rendendoli disponibili per applicazioni aziendali specifiche.

La piattaforma IoT universale di HPE garantisce un unico allineamento standard, indipendenza da dispositivi e fornitori e astrazione della rete sottostante.
La piattaforma incorpora caratteristiche IoT chiave che garantiscono facilità di utilizzo per gli sviluppatori finali e consentono di ridurre i costi dell'infrastruttura con un TCO[15] ridotto comprovato e a un'elevata scalabilità.

La piattaforma IoT universale di HPE offre la caratteristica unica di un singolo punto di responsabilità e di un singolo fornitore di riferimento per gestire e connettere gli insiemi eterogenei di dispositivi IoT e consentire il funzionamento di applicazioni verticali su dispositivi machine-to-machine (M2M) da diversi paesi.

Attualmente l'obiettivo è di acquisire dati, convalidarli, arricchirli attraverso analisi, abbinarli ad altre fonti e quindi inviarli alle applicazioni che consentono ai clienti di ottenere valore aziendale da questi servizi.

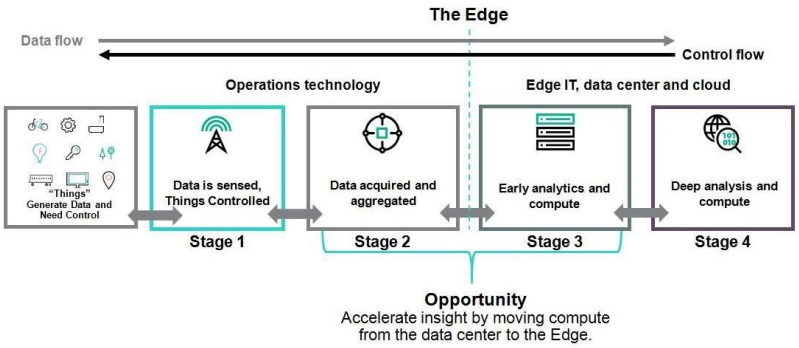


Figura 4.21: Infografica di HPE IoT

[15] **Total Cost of Ownership (TCO)**: In italiano costo totale di proprietà o costo totale di possesso, è un approccio sviluppato da Gartner nel 1987, utilizzato per calcolare tutti i costi del ciclo di vita di un'apparecchiatura informatica IT, per l'acquisto, l'installazione, la gestione, la manutenzione e il suo smaltimento (RAEE).

3.2.10. Stoorm5

3.2.10.1.Panoramica

Stoorm5 permette la creazione di applicazioni IoT con semplici passaggi, la tecnologia IoT è tutta dentro la piattaforma.

Tramite le API i dispositivi vengono connessi a Stoorm5 e controllati esternamente, con sistemi software esistenti o con nuove app scritte con qualsiasi linguaggio di programmazione.

Figura 4.22: Spiegazione della piattaforma Stoorm5

I dati inviati a Stoorm5 vengono memorizzati e resi disponibili in real-time su database SQL o NoSQL a scelta del cliente.
I dati sono memorizzati con un'innovativa data model che permette l'analisi con tecniche di Business Intelligence e Big Data.

Una volta connessi gli oggetti a Stoorm5 è possibile monitorarli e comandarli direttamente dalla piattaforma, cambiare parametri, inviare comandi e fare debug remoto in caso di eventi imprevisti.

3.3. Conclusioni

La piattaforma Ubidots, la quale possiede un'interfaccia altamente "*user friendly*" è la più adatta alle necessità del progetto essendo anche completamente gratuita nella versione "*Education*", che offre per un periodo illimitato di tempo un set quasi completo di strumenti.

È semplice collegare l'hardware al cloud Ubidots con più di 200 librerie, SDK e tutorial comprovati dall'utente per guidare la tua integrazione su HTTP, MQTT, TCP, UDP o mediante analisi di protocolli personalizzati/industriali.

L'infrastruttura delle serie temporali di Ubidots è ottimizzata per ricevere, calcolare e restituire milioni di punti dati ogni secondo in tutto il mondo. E, con due anni standard di conservazione dei dati a rotazione, Ubidots Cloud offre agli integratori e agli innovatori un posto dove archiviare ed estrarre i dati per ulteriori approfondimenti e sovrapposizioni analitiche come il rilevamento di anomalie o la manutenzione predittiva.

Utilizzando gli strumenti di sviluppo delle applicazioni *point-and-click* di Ubidots, è possibile creare *dashboard* in tempo reale per analizzare i dati e controllare i dispositivi. Si possono visualizzare i dati con i grafici azionari Ubidots, le tabelle, gli indicatori, le mappe, le metriche e i widget di controllo o è possibile svilupparne utilizzando la tela HTML e un codice personalizzato.
È inoltre possibile condividere i dati tramite link pubblici o incorporando *dashboard* o widget in applicazioni Web e mobili private.

Il valore aggiunto di Ubidots nei tempi di sviluppo e nei costi risparmiati aumenta solo se combinato con la sua usabilità. Con un'architettura di base focalizzata sull'efficienza dei dati e una coinvolgente UX ("*user experience*"), gli utenti Ubidots possono connettersi, costruire e distribuire applicazioni IoT cloud con facilità, lasciando Ubidots per gestire il cloud e l'infrastruttura UX dell'utente finale.

Le soluzioni connesse a Internet non sono facili, nessuna tecnologia emergente lo è, ma Ubidots si concentra sugli input chiave per fornire gli strumenti giusti e in modo tempestivo affinché la soluzione abbia successo.
Per questi motivi la piattaforma Ubidots STEM è stata utilizzata in questo progetto.

CAPITOLO 4:
PYTHON

In questo capitolo verrà introdotto il linguaggio Python utilizzato per la scrittura del programma per la Raspberry.

Verranno spiegate le motivazioni della scelta di tale linguaggio, verrà effettuata una spiegazione generale del linguaggio e verranno spiegate le librerie che saranno poi utilizzate nel codice.

4.1. Introduzione

Python è un linguaggio di programmazione potente e facile da imparare. Ha strutture dati efficienti di alto livello e un approccio semplice ma efficace alla programmazione orientata agli oggetti. L'elegante sintassi e la digitazione dinamica di Python, insieme alla sua natura, lo rendono un linguaggio ideale per lo scripting e lo sviluppo rapido di applicazioni in molte aree della maggior parte delle piattaforme.

L'interprete Python e la vasta libreria standard sono disponibili gratuitamente in formato sorgente o binario per tutte le principali piattaforme dal sito Web Python, https://www.python.org/, e possono essere distribuiti liberamente. Lo stesso sito contiene anche distribuzioni e puntatori a molti moduli, programmi e strumenti Python di terze parti gratuiti e documentazione aggiuntiva.

L'interprete Python può essere facilmente esteso con nuove funzioni e tipi di dati implementati in C o C++ (o altri linguaggi richiamabili da C).

Python è adatto anche come linguaggio di estensione per applicazioni personalizzabili.

4.2. Client Paho MQTT

Il nucleo della libreria client è la classe client che fornisce tutte le funzioni per pubblicare messaggi e iscriversi ai *topic*.
È possibile utilizzare la classe client come istanza, all'interno di una classe o tramite sottoclasse. Il flusso di utilizzo generale è il seguente:

- Creare un'istanza client;
- Connettersi a un broker utilizzando una delle funzioni connect();
- Chiamare una delle funzioni loop() per mantenere il flusso del traffico di rete con il broker;
- Utilizzare subscribe() per iscriverti a un argomento e ricevere messaggi;
- Utilizzare publish() per pubblicare messaggi nel broker;
- Utilizzare disconnect() per disconnettersi dal broker;

I callback verranno chiamati per consentire all'applicazione di elaborare gli eventi secondo necessità.
Queste funzioni e i relativi callback verranno descritti di seguito.

4.2.1. Client/Reinitialise

4.2.1.1. Client

```
Client(client_id= "", clean_session=True, userdata=None,
protocol=MQTTv311, transport= "tcp")
```

Il costrutto Client() accetta i seguenti argomenti:

- **Client_id,** stringa univoca utilizzata durante la connessione al broker.
  Se client_id ha lunghezza zero o "*None*", ne verrà generato uno a caso. In questo caso il parametro clean_session deve essere True;
- **Clean_session,** valore booleano che determina il tipo di client.
  Se **True**, il broker rimuoverà tutte le informazioni su questo client quando si disconnette.
  Se **False**, il client è durevole e le informazioni sull'abbonamento e i messaggi in coda verranno conservati quando il client si disconnette.
  Si noti che un client non scarterà mai i propri messaggi in uscita al momento della disconnessione.
  La chiamata a connect() o reconnect() provocherà il rinvio dei messaggi.
  È necessario utilizzare reinitialise() per ripristinare un client al suo stato originale.
- **Userdata,** dati definiti dall'utente di qualsiasi tipo passati come parametro userdata ai callback.
  Potrebbe essere aggiornato in un secondo momento con la funzione user_data_set ().
- **Protocol,** la versione del protocollo MQTT da utilizzare per questo client.
  Può essere MQTTv31 o MQTTv311.
- **Trasport,** impostato su "websocket" per inviare MQTT su WebSocket.
  Lasciare l'impostazione predefinita di "tcp" per utilizzare il TCP non elaborato.

```
import paho.mqtt.client as mqtt


mqttc = mqtt.Client()
```

Figura 4.1: Esempio di costrutto client

4.2.1.2. Reinitialise

```
reinitialise(client_id= "", clean_session=True, userdata=None)
```

La funzione reinitialise() reimposta il client al suo stato iniziale come se fosse stato appena creato. Prende gli stessi argomenti del costrutto Client().

4.2.1.3. Funzioni opzionali

Queste funzioni rappresentano opzioni che possono essere impostate sul client per modificarne il comportamento.
Nella maggior parte dei casi questo deve essere fatto prima di connettersi a un broker.

MAX_INFLIGHT_MESSAGES_SET

```
max_inflight_messages_set(self, inflight)
```

Imposta il numero massimo di messaggi (con QoS> 0) che possono essere contemporaneamente nello stesso flusso di rete.
Il valore predefinito è 20 e l'aumento di questo valore consumerà più memoria ma può anche aumentare la velocità effettiva.

MAX_QUEUED_MESSAGES_SET

```
max_queued_messages_set(self, queue_size)
```

Imposta il numero massimo di messaggi in uscita (con QoS> 0) che può essere in sospeso nella coda dei messaggi in uscita.
Il valore predefinito è 0 (0 significa illimitato). Quando la coda è piena, tutti i messaggi in uscita verranno eliminati.

MESSAGE_RETRY_SET

```
message_retry_set(retry)
```

Imposta il tempo in secondi prima che venga riprovato un messaggio (con QoS> 0) nel caso in cui il broker non risponda.
L'impostazione predefinita è cinque secondi e in genere non dovrebbe essere modificata.

WS_SET_OPTIONS

```
ws_set_options(self, path= "/mqtt", headers=None)
```

Imposta le opzioni di connessione al websocket.

Queste opzioni verranno utilizzate solo se transport = "websocket" è stato passato al costrutto Client(). Gli argomenti della funzione sono i seguenti:

- **Path,** Il percorso mqtt da utilizzare sul broker.
- **Headers,** un dizionario che specifica un elenco di intestazioni aggiuntive che devono essere aggiunte alle intestazioni standard del websocket o un *callable* che accetta le normali intestazioni del websocket e restituisce un nuovo dizionario con un set di intestazioni per connettersi al broker.

Deve essere chiamato prima di connect().

TLS_SET

```
tls_set(ca_certs=None, certfile=None, keyfile=None,
cert_reqs=ssl.CERT_REQUIRED,tls_version=ssl.PROTOCOL_TLS, ciphers=None)
```

Configura le opzioni di crittografia e autenticazione di rete. Abilita il supporto SSL/TLS e accetta i seguenti argomenti:

- **Ca_certs,** un percorso stringa per i file di certificato dell'autorità di certificazione che devono essere considerati affidabili da questo client.
  Se questa è l'unica opzione fornita, il client funzionerà in modo simile a un browser web. Ciò significa che richiederà al broker di avere un certificato firmato dalle autorità di certificazione in ca_certs e comunicherà utilizzando TLS v1, ma non tenterà alcuna forma di autenticazione.
  Questo fornisce la crittografia di rete di base ma potrebbe non essere sufficiente a seconda della configurazione del broker.
  Per impostazione predefinita, su Python 2.7.9+ o 3.4+, viene utilizzata l'autorità di certificazione predefinita del sistema, mentre nelle versioni precedenti questo parametro è obbligatorio;
- **Certfile/Keyfile,** stringhe che puntano rispettivamente al certificato client codificato PEM e alle chiavi private.
  Se questi argomenti non sono "*None*", verranno utilizzati come informazioni client per l'autenticazione basata su TLS. Il supporto per questa funzione dipende dal broker.
  Se uno di questi file è crittografato e necessita di una password per decrittografarlo, Python chiederà la password attraverso la riga di comando. Al momento non è possibile definire un callback per fornire la password;

- **Cert_reqs,** definisce i requisiti del certificato che il client impone al broker.
  Per impostazione predefinita, questo è ssl.CERT_REQUIRED, il che significa che il broker deve fornire un certificato;
- **Tls_version,** specifica la versione del protocollo SSL/TLS da utilizzare.
  Per impostazione predefinita (se la versione di Python lo supporta) viene rilevata la versione TLS più alta. Se non disponibile, viene utilizzato TLS v1.
  Le versioni precedenti (tutte le versioni che iniziano con SSL) sono possibili ma non consigliate a causa di possibili problemi di sicurezza;
- **Ciphers,** una stringa che specifica quali codici di crittografia sono consentiti per questa connessione , viene assegnato "*None*"per utilizzare i valori predefiniti.

Deve essere chiamato prima di connect().

TLS_SET_CONTEXT

```
tls_set_context(context=None)
```

Configura la crittografia di rete e il contesto di autenticazione. Abilita il supporto SSL/TLS e richiede il seguente argomento:

- **Context,** un oggetto ssl.SSLContext.
  Per impostazione predefinita, questo è dato da ssl.create_default_context(), se disponibile (aggiunto in Python 3.4).

Deve essere chiamato prima di connect().

TLS_INSECURE_SET

```
tls_insecure_set(value)
```

Configura la verifica del nome host del server nel certificato del server.

Se il valore è impostato su **True**, è impossibile garantire che l'host a cui ci si sta connettendo non stia impersonando il proprio server.
Ciò può essere utile nel test iniziale del server, ma consente ad una terza parte malintenzionata di rappresentare il tuo server, ad esempio tramite lo spoofing DNS.

Impostando il valore su True significa che non ha senso utilizzare la crittografia.

Deve essere chiamato prima di connect() e dopo tls_set() o tls_set_context().

ENABLE_LOGGER

```
enable_logger(logger=None)
```

Abilita la registrazione utilizzando il pacchetto di registrazione standard di Python. Questo può essere usato contemporaneamente al metodo di callback on_log.

Se viene specificato logger, verrà utilizzato l'oggetto logging.logger, altrimenti ne verrà creato uno automaticamente.

I livelli di registrazione di Paho vengono convertiti in livelli standard in base alla seguente mappatura:

Paho	logging
MQTT_LOG_ERR	logging.ERROR
MQTT_LOG_WARNING	logging.WARNING
MQTT_LOG_NOTICE	logging.INFO *(no direct equivalent)*
MQTT_LOG_INFO	logging.INFO
MQTT_LOG_DEBUG	logging.DEBUG

DISABLE_LOGGER

```
disable_logger()
```

Disabilita la registrazione utilizzando il pacchetto di registrazione standard di Python. Ciò non ha alcun effetto sul callback on_log.

USERNAME_PW_SET

```
username_pw_set(username, password=None)
```

Imposta un nome utente e, facoltativamente, una password per l'autenticazione del broker.
Deve essere chiamato prima di connect().

USER_DATA_SET

```
user_data_set(userdata)
```

Imposta i dati dell'utente privato che verranno passati ai callback quando vengono generati eventi.

WILL_SET

```
will_set(topic, payload=None, qos=0, retain=False)
```

Imposta una "volontà" da inviare al broker. Se il client si disconnette senza chiamare disconnect(), il broker pubblicherà il messaggio per suo conto.

Questa funzione accetta i seguenti argomenti:

- **Topic,** argomento su cui dovrebbe essere pubblicato il messaggio di volontà;
- **Payload,** messaggio da inviare come testamento;
  Se non specificato, o impostato su "*None*" verrà utilizzato un messaggio di lunghezza zero come testamento. Il passaggio di un *int* o *float* comporterà la conversione del payload in una stringa che rappresenta quel numero.
  Se si desidera inviare un vero *int/float* è necessario utilizzare struct.pack() per creare il payload richiesto;
- **QoS,** qualità del livello di servizio da utilizzare per la volontà;
- **Retain,** se impostato su **True**, il messaggio di volontà verrà impostato come "ultimo bene noto"/messaggio conservato per l'argomento, verrà dunque conservato fino alla prossima connessione.

Genera un ValueError se qos non è 0, 1 o 2 o se l'argomento è "*None*" o ha lunghezza della stringa pari a zero.

RECONNECT_DELAY_SET

```
reconnect_delay_set(min_delay=1, max_delay=120)
```

Il client riproverà automaticamente la connessione. Tra ogni tentativo attenderà un numero di secondi tra min_delay e max_delay.

Quando la connessione viene persa il tentativo di riconnessione iniziale viene ritardato di min_delay secondi, venendo poi raddoppiato tra i tentativi successivi fino a max_delay.

Il ritardo viene reimpostato su min_delay al termine della connessione (ad esempio, quando viene ricevuto il CONNACK).

4.2.2. Connect/reconnect/disconnect

4.2.2.1. Connessione

```
connect(host, port=1883, keepalive=60, bind_address= "")
```

La funzione connect() collega il client a un broker. Questa è una funzione di blocco e prende i seguenti argomenti:

- **Host,** nome host o l'indirizzo IP del broker remoto;
- **Port,** porta di rete dell'host server a cui connettersi. Il valore predefinito è 1883. Notare che la porta predefinita per MQTT su SSL/TLS è 8883, quindi se si utilizza tls_set () o tls_set_context (), potrebbe essere necessario fornire manualmente la porta;
- **Keepalive,** periodo massimo in secondi consentito tra le comunicazioni con il broker. Se non vengono scambiati altri messaggi, questo controlla la velocità con cui il client invierà messaggi ping al broker;
- **Bind_address,** indirizzo IP di un'interfaccia di rete locale a cui associare questo client, presupponendo che esistano più interfacce.

```
mqttc.connect("mqtt.eclipse.org")
```

Figura 4.2: Esempio di connessione

4.2.2.2. Connessione asincrona

```
connect_async(host, port=1883, keepalive=60, bind_address= "")
```

Utilizzata in combinazione con loop_start() per connettersi in modo non bloccante.
La connessione non verrà completata fino a quando non viene chiamato loop_start().

4.2.2.3. Connessione srv

```
connect_srv(domain, keepalive=60, bind_address= "")
```

Permette di connettersi ad un broker utilizzando una ricerca DNS SRV per ottenere l'indirizzo del broker. Accetta i seguenti argomenti:

- **Domain,** il dominio DNS per cercare i record SRV. Se Nessuno, prova a determinare il nome di dominio locale.
- Vedi connect() per una descrizione degli argomenti **keepalive** e **bind_address**.

```
mqttc.connect_srv( "eclipse.org" )
```

Figura 4.3: Esempio di SRV Connect

4.2.2.4. Riconnettersi

```
reconnect()
```

Consente di riconnettersi ad un broker utilizzando i dettagli forniti in precedenza. È necessario aver chiamato connect() prima di chiamare questa funzione.

4.2.2.5. Disconnettersi

```
disconnect()
```

Disconnette dal broker in modo pulito. L'uso di disconnect() non comporterà l'invio di un messaggio di volontà da parte del broker.

Disconnect() non attenderà l'invio di tutti i messaggi in coda per garantire che tutti i messaggi vengano recapitati, per fare ciò dovrebbe essere utilizzato wait_for_publish () da MQTTMessageInfo.

4.2.3. Network loop

Queste funzioni sono la forza trainante del client.
Se non vengono chiamati, i dati di rete in entrata non verranno elaborati e i dati di rete in uscita potrebbero non essere inviati in modo tempestivo. Esistono quattro opzioni per la gestione del loop di rete e di seguito vengono descritte tre di queste.

4.2.3.1. Ciclo

```
loop(timeout=1.0, max_packets=1)
```

Chiama regolarmente per elaborare gli eventi di rete. Questa chiamata attende in select() fino a quando il socket di rete non è disponibile per la lettura o la scrittura, se appropriato, quindi gestisce i dati in entrata/in uscita.

Questa funzione si blocca per un massimo di secondi di timeout, il quale non deve superare il valore keepalive per il client o quest'ultimo verrà disconnesso dal broker. L'argomento max_packets è obsoleto e deve essere lasciato non impostato.

```
run = True

while run:
    mqttc.loop()
```

Figura 4.4: Esempio di Loop

4.2.3.2. Ciclo inizio/fine

```
loop_start()

loop_stop(force=False)
```

Queste funzioni implementano un'interfaccia filettata al loop di rete.

Chiamando loop_start () una volta, prima o dopo connect(), viene eseguito un thread in background per chiamare loop() automaticamente. Questo libera il thread principale per altri lavori che potrebbero bloccare.

Questa chiamata gestisce anche la riconnessione al broker. Viene chiamato loop_stop() per interrompere il thread in background. L'argomento force è attualmente ignorato.

```
mqttc.connect("mqtt.eclipse.org")
mqttc.loop_start()

while True:
    temperature = sensor.blocking_read()
    mqttc.publish("paho/temperature", temperature)
```

Figura 4.5: Esempio di Loop Start/Stop

4.2.3.3. Ciclo infinito

```
loop_forever(timeout=1.0, max_packets=1, retry_first_connection=False)
```

Questa è una forma di blocco del loop di rete e non tornerà fino a quando il client non chiama disconnect(). Gestisce automaticamente la riconnessione.

Ad eccezione del primo tentativo di connessione, quando si utilizza connect_async, è necessario utilizzare retry_first_connection = True per fare in modo che venga riprovata la prima connessione.

Questo potrebbe portare a situazioni in cui il client continua a connettersi ad un host inesistente senza errori.

Gli argomenti timeout e max_packets sono obsoleti e devono essere lasciati non impostati.

4.2.4. Publishing

Funzione che invia un messaggio dal client al broker.

4.2.4.1. Pubblicare

```
publish(topic, payload=None, qos=0, retain=False)
```

Ciò provoca l'invio di un messaggio al broker e successivamente dal broker a tutti i client che abbonano argomenti corrispondenti. Prende i seguenti argomenti:

- **Topic,** argomento su cui dovrebbe essere pubblicato il messaggio;
- **Payload,** messaggio da inviare come testamento;
  Se non specificato, o impostato su "*None*" verrà utilizzato un messaggio di lunghezza zero come testamento. Il passaggio di un *int* o *float* comporterà la conversione del payload in una stringa che rappresenta quel numero.
  Se si desidera inviare un vero *int/float* è necessario utilizzare struct.pack() per creare il payload richiesto;
- **QoS,** qualità del livello di servizio da utilizzare per la volontà;
- **Retain,** se impostato su **True**, il messaggio di volontà verrà impostato come "ultimo bene noto"/messaggio conservato per il topic, verrà dunque conservato fino alla prossima connessione.

Restituisce un MQTTMessageInfo che espone i seguenti attributi e metodi:

- **Rc,** il risultato della pubblicazione.
  Potrebbe essere MQTT_ERR_SUCCESS a indicare il successo, MQTT_ERR_NO_CONN se il client non è attualmente connesso, oppure MQTT_ERR_QUEUE_SIZE quando max_queued_messages_set viene utilizzato per indicare che il messaggio non è né in coda né inviato.
- **Mid,** è l'ID del messaggio per la richiesta di pubblicazione.
  Il valore mid può essere utilizzato per tenere traccia della richiesta di pubblicazione controllando l'argomento mid nel callback on_publish() se è definito.
- **Wait_for_publish(),** si bloccherà fino alla pubblicazione del messaggio.
  Aumenterà ValueError se il messaggio non è in coda (rc == MQTT_ERR_QUEUE_SIZE).
- **Is_published,** restituisce **True** se il messaggio è stato pubblicato.
  Aumenterà ValueError se il messaggio non è in coda (rc == MQTT_ERR_QUEUE_SIZE).

4.2.5. Subscribe/Unsubscribe

4.2.5.1. Iscriversi

```
subscribe(topic, qos=0)
```

Questa funzione permette al client di iscriversi ad uno o più topic e può essere chiamata in tre modi diversi:

- **Stringa e numero intero semplici:**
  es.: Subscribe("my/topic", 2)
  - **Topic,** una stringa che specifica l'argomento di iscrizione a cui iscriversi.
  - **QoS,** il livello di qualità del servizio desiderato per l'abbonamento. Il valore predefinito è 0.
- **Tupla di stringhe e numeri interi:**
  es.: Subscribe(("my/topic", 1))
  - **Topic,** una tupla di (topic, qos).
    Sia l'argomento che il qos devono essere presenti nella tupla.
  - **QoS,** non usato.
- **Elenco di tuple di stringhe e numeri interi**
  es.: Subscribe([("my/topic", 0), ("another/topic", 2)])
  Ciò consente l'iscrizione a più argomenti in un singolo comando SUBSCRIBE, che è più efficiente dell'uso di più chiamate per subscribe().
  - **Topic,** un elenco di tuple di formato (topic, qos).
    Sia l'argomento che il qos devono essere presenti in tutte le tuple.
  - **QoS,** non usato.

La funzione restituisce una tupla (risultato, mid), dove il risultato è MQTT_ERR_SUCCESS per indicare il successo o MQTT_ERR_NO_CONN/*None* se il client non è attualmente connesso.

Mid è l'ID del messaggio per la richiesta di iscrizione. Il valore mid può essere utilizzato per tenere traccia della richiesta di iscrizione verificando l'argomento mid nel callback on_subscribe() (il quale verrà descritto in seguito) se è definito.

Genera un ValueError se qos non è 0, 1 o 2 o se il topic è *None*, ha lunghezza stringa zero o se non è una stringa, tupla o elenco.

4.2.5.2. Annullare l'iscrizione

```
unsubscribe(topic)
```

Annulla l'iscrizione al client da uno o più topic.

- **Topic,** una singola stringa o un elenco di stringhe che sono gli argomenti di sottoscrizione da cui annullare l'iscrizione.

Restituisce una tupla (risultato, mid), dove risultato è MQTT_ERR_SUCCESS per indicare il successo, oppure MQTT_ERR_NO_CONN/*None* se il client non è attualmente connesso.

Mid è l'ID del messaggio per la richiesta di annullamento dell'iscrizione. Il valore mid può essere utilizzato per tenere traccia della richiesta di annullamento dell'iscrizione verificando l'argomento mid nel callback on_unsubscribe() (il quale verrà descritto in seguito) se è definito.

Genera un ValueError se l'argomento è *None* o ha lunghezza stringa zero o non è una stringa o un elenco.

4.2.6. Callbacks

4.2.6.1. Richiamo in connessione

```
on_connect(client, userdata, flags, rc)
```

Chiamato quando il broker risponde alla nostra richiesta di connessione. Il richiamo accetta i seguenti argomenti:

- **Client,** l'istanza client per questo callback;
- **Userdata,** i dati dell'utente privato come impostato in Client() o user_data_set();
- **Flags,** bandiere di risposta inviate dal broker;
- **Rc,** il risultato della connessione.
  Il valore di rc indica il successo o l'insuccesso della connesione:
  - 0: connessione riuscita;
  - 1: connessione rifiutata - versione protocollo errata;
  - 2: connessione rifiutata - identificativo client non valido;
  - 3: connessione rifiutata - server non disponibile;
  - 4: connessione rifiutata - nome utente o password errati;
  - 5: connessione rifiutata - non autorizzata.

```
def on_connect(client, userdata, flags, rc):
    print( "Connection returned result: " +connack_string(rc))

mqttc.on_connect = on_connect
...
```

Figura 4.6: Esempio di On_Connect

4.2.6.2. Richiamo in disconnessione

```
on_disconnect(client, userdata, rc)
```

Chiamato quando il client si disconnette dal broker.

- **Client,** l'istanza client per questo callback;
- **Userdata,** i dati dell'utente privato come impostato in Client() o user_data_set();
- **Rc,** il risultato di disconnessione.

Se MQTT_ERR_SUCCESS (0) allora il callback è stato chiamato in risposta a una chiamata disconnect().

Se assume qualsiasi altro valore significa che la disconnessione è stata inaspettata, ad esempio potrebbe essere causata da un errore di rete.

```
def on_disconnect(client, userdata, rc):
    if rc != 0:
        print("Unexpected disconnection. ")

mqttc.on_disconnect = on_disconnect
...
```

Figura 4.7: Esempio di On_Disconnect

4.2.6.3. Richiamo in messaggio

```
on_message(client, userdata, message)
```

Viene chiamato quando è stato ricevuto un messaggio su un topic a cui il client si iscrive e il messaggio non corrisponde a un callback del filtro argomenti esistente.

Si può utilizzare message_callback_add() per definire un callback che verrà chiamato per filtri argomenti specifici. on_message fungerà da fallback quando nessuno corrisponde.

- **Client,** l'istanza client per questo callback;
- **Userdata,** i dati dell'utente privato come impostato in Client() o user_data_set();
- **Message,** un'istanza di MQTTMessage.
  Questa è una classe con argomento membri, payload, qos, trattenere.

```
def on_message(client, userdata, message):
    print("Received message " + str(message.payload) + " on topic "
        + message.topic + " with QoS " + str(message.qos))

mqttc.on_message = on_message
...
```

Figura 4.8: Esempio di On_Message

4.2.6.4. Aggiunta di richiamo in messaggio

Questa funzione consente di definire callback che gestiscono i messaggi in arrivo per filtri di iscrizione specifici, anche con caratteri jolly.

Ciò consente, ad esempio, di abbonarsi a sensori/# e avere un callback per gestire sensori/temperatura e un altro per gestire sensori/umidità.

```
message_callback_add(sub, callback)
```

- **Sub,** filtro di abbonamento a cui corrispondere per questo callback.
  È possibile definire un solo callback per sottostringa letterale;
- **Callback,** callback da usare.
  Assume la stessa forma del callback on_message.

Se si utilizza message_callback_add() e on_message, solo i messaggi che non corrispondono a un filtro specifico per l'abbonamento verranno passati al callback on_message.

Se più sottotitoli corrispondono a un topic, verrà richiamato ciascun callback (ad es. Sub-sensori/# e sub +/umidità entrambi corrispondono a un messaggio con un argomento sensori/umidità, quindi entrambi i callback gestiranno questo messaggio).

4.2.6.5. Rimozione di richiamo in messaggio

Rimuove un callback specifico per topic/iscrizione precedentemente registrato utilizzando message_callback_add().

```
message_callback_remove(sub)
```

- **Sub,** filtro di abbonamento da rimuovere.

4.2.6.6. Richiamo in pubblicazione

```
on_publish(client, userdata, mid)
```

Viene chiamato quando un messaggio che doveva essere inviato utilizzando la chiamata publish() ha completato la trasmissione al broker.

Per i messaggi con livelli QoS 1 e 2, ciò significa che sono state completate le opportune handshake. Per QoS 0, ciò significa semplicemente che il messaggio ha lasciato il client.

La variabile mid corrisponde alla variabile mid restituita dalla corrispondente chiamata publish(), per consentire il monitoraggio dei messaggi in uscita.

Questo callback è importante perché anche se la chiamata publish() restituisce esito positivo, non significa sempre che il messaggio è stato inviato.

4.2.6.7. Richiamo in iscrizione

```
on_subscribe(client, userdata, mid, granted_qos)
```

Viene chiamato quando il broker risponde a una richiesta di iscrizione.

La variabile mid corrisponde alla variabile mid restituita dalla chiamata subscribe() corrispondente. La variabile grant_qos è un elenco di numeri interi che forniscono il livello QoS concesso dal broker per ciascuna delle diverse richieste di sottoscrizione.

4.2.6.8. Richiamo in annullamento di iscrizione

```
on_unsubscribe(client, userdata, mid)
```

Viene chiamato quando il broker risponde a una richiesta di annullamento dell'iscrizione.

La variabile mid corrisponde alla variabile mid restituita dalla corrispondente chiamata unsubscribe().

4.2.6.9. Richiamo in log

```
on_log(client, userdata, level, buf)
```

Viene chiamato quando il client dispone delle informazioni di registro ed è necessario definirla per consentire il debug.

La variabile level indica la gravità del messaggio e sarà una di MQTT_LOG_INFO, MQTT_LOG_NOTICE, MQTT_LOG_WARNING, MQTT_LOG_ERR e MQTT_LOG_DEBUG.

Questo può essere usato contemporaneamente alla registrazione Python standard, che può essere abilitata tramite il metodo enable_logger.

4.2.6.10. Richiamo in apertura socket

```
on_socket_open(client, userdata, sock)
```

Chiamato quando il socket è stato aperto.
È necessario utilizzare questa funzione per registrare il socket con un loop di eventi esterno per la lettura.

4.2.6.11. Richiamo in chiusura socket

```
on_socket_close(client, userdata, sock)
```

Chiamato quando il socket sta per essere chiuso.
È necessario utilizzare questa funzione per annullare la registrazione di un socket da un loop di eventi esterno per la lettura.

4.2.6.12. Richiamo in registrazione di socket

```
on_socket_register_write(client, userdata, sock)
```

Chiamato quando un'operazione di scrittura sul socket non è riuscita perché si sarebbe bloccata (Es.: buffer di output pieno).
È necessario utilizzare questa funzione per registrare il socket con un loop di eventi esterno per la scrittura.

4.2.6.13. Richiamo in annullamento di registrazione di socket

```
on_socket_unregister_write(client, userdata, sock)
```

Chiamato quando un'operazione di scrittura sul socket ha avuto esito positivo dopo un precedente fallimento.

Viene utilizzata questa funzione per annullare la registrazione del socket da un loop di eventi esterno per la scrittura.

```
import paho.mqtt.client as mqtt
# The callback for when the client receives a CONNACK response from the
# server.
def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))

    # Subscribing in on_connect() means that if we lose the connection
and
    # reconnect then subscriptions will be renewed.
    client.subscribe("$SYS/#")

# The callback for when a PUBLISH message is received from the server.
def on_message(client, userdata, msg):
    print(msg.topic+ " " +str(msg.payload))

client = mqtt.Client()

client.on_connect = on_connect

client.on_message = on_message

client.connect("mqtt.eclipse.org", 1883, 60)

# Blocking call that processes network traffic, dispatches callbacks and
# handles reconnecting.
# Other loop*() functions are available that give a threaded interface
and
# a manual interface.
client.loop_forever()
```

Figura 4.9: Esempio di iscrizione all'albero degli argomenti del broker $SYS e stampa dei messaggi risultanti

4.3. Librerie

Le librerie sono insiemi di routine e funzioni scritte che svolgono un determinato compito, che può essere richiamato a seconda delle necessità.

Innanzitutto possiamo considerare la libreria come un insieme di moduli.
Ogni modulo contiene delle istruzioni e definizioni semplici. L'accorpamento di vari moduli, quindi di istruzione codice, costituisce una libreria.
Spesso i moduli sono già stati scritti da altri sviluppatori, e non c'è bisogno di ripartire da capo ogni volta. Il loro scopo è quello di semplificare le attività, aiutando gli sviluppatori a scrivere solo poche righe anziché una grande quantità di comandi.

Il codice delle librerie richiama classi e metodi che normalmente definiscono operazioni specifiche in un'area del dominio.
Ad esempio, ci sono alcune librerie di matematica che possono far sì che lo sviluppatore chiami semplicemente la funzione senza ripetere l'implementazione di come funziona un algoritmo.

Verranno discusse quindi le librerie utilizzate nel programma del progetto.

4.3.1. Requests

Fare una richiesta con Requests è molto semplice. Si inizia importando il modulo:

```
import requests
```

Si può ad esempio ottenere una pagina web. Per questo esempio, prendiamo la cronologia pubblica di GitHub:

```
r=request.get("https://api.github.com/events")
```

Ora, si ha un oggetto Response chiamato r. È possibile ottenere tutte le informazioni di cui si ha bisogno da questo oggetto.

La semplice API delle richieste indica che tutte le forme di richiesta HTTP sono altrettanto ovvie. Ad esempio, ecco come si effettua una richiesta POST HTTP:

```
r=request.post("https://httpbin.org/post", data={"key" : "value"})
```

Figura 4.10: Esempio di POST HTTP

È possibile anche effettuare tutti gli atri tipi di richiesta HTTP: PUT, DELETE, HEAD e OPTIONS; e sono tutti altrettanto semplici:

```
r=request.put("https://httpbin.org/put", data={"key" : "value"})
r=request.delete("https://httpbin.org/delete")
r=request.head("https://httpbin.org/get")
r=request.options("https://httpbin.org/get")
```

Figura 4.11: Funzioni di PUT,DELETE,HEAD,OPTIONS HTTP

4.3.1.1. Passare i parametri negli URL

Spesso è necessario inviare una sorta di dati nella stringa di query dell'URL.

Se si sta costruendo l'URL manualmente, questi dati verrebbero forniti come coppie chiave/valore nell'URL dopo un punto interrogativo (Es.: httpbin.org/get?key=val).

La libreria Requests permette di fornire questi argomenti come un dizionario di stringhe, usando l'argomento della parola chiave params.

Ad esempio, se si desidera passare key1 = value1 e key2 = value2 a httpbin.org/get, si può utilizzare il seguente codice:

```
payload={"key1" : "value1", "key2" : "value2"}
r=request.get("https://httpbin.org/get", params=payload)
```

Figura 4.12: Esempio di codice per i parametri nell'URL

È possibile vedere che l'URL è stato codificato correttamente stampandolo:

```
print(r.url)
```

E si otterrà un output del tipo:

```
https://httpbin.org/get?key2=value2&key1=value1
```

Qualsiasi chiave del dizionario il cui valore è *None* non verrà aggiunta alla stringa di query dell'URL. Si può anche passare un elenco di elementi come valore (Esempio in figura 4.13):

```
payload={"key1" : "value1", "key2" : ["value2", "value3"]}

r=request.get("https://httpbin.org/get", params=payload)
print(r.url)
# https://httpbin.org/get?key1=value1&key2=value2&key2=value3
```

Figura 4.13: Esempio di passaggio di elenco di elementi

4.3.1.2. Contenuto della risposta

Attraverso la libreria Requests si è in grado di leggere il contenuto della risposta del server.
Si consideri di nuovo la sequenza temporale di GitHub:

```
import requests


r=request.get("https://api.github.com/events")
r.text
# u '[{ "repository": { "open_issues": 0, "URL": "https: //github.com /
...
```

Le richieste decodificheranno automaticamente il contenuto dal server. La maggior parte dei set di caratteri Unicode viene decodificata senza soluzione di continuità.

Quando si effettua una richiesta, Requests fa ipotesi ponderate sulla codifica della risposta in base alle intestazioni HTTP. La codifica del testo indovinata da Requests viene utilizzata quando si accede a r.text.

Si può scoprire quali richieste di codifica la libreria stia usando e modificarle usando la proprietà r.encoding:

```
r.encoding


# "Utf-8"
r.encoding= "ISO-8859-1"
```

Figura 4.14: Esempio di encoding

Se si modifica la codifica, Requests utilizzerà il nuovo valore di r.encoding ogni volta che si chiama r.text.

Questo è possibile farlo in qualsiasi situazione, ad esempio ogni volta in cui si può applicare una logica speciale per capire quale sarà la codifica del contenuto.

HTML e XML hanno la capacità di specificare la loro codifica nel loro corpo. In situazioni come questa, si dovrebbe utilizzare r.content per trovare la codifica e quindi impostare r.encoding.
Questo permetterà di usare r.text con la codifica corretta.

Le richieste utilizzeranno anche codifiche personalizzate nel caso in cui siano necessarie. Se è stata creata una codifica personalizzata ed è stata registrata con il modulo codec, è possibile usare il nome codec come valore di r.encoding e Requests gestirà la decodifica.

4.3.1.3. Contenuto di risposta binaria

Si può anche accedere al corpo della risposta come byte, per richieste non testuali:

```
r.content
# b '[{ "repository": { "open_issues": 0, "URL": "https: //github.com /
...
```

Le codifiche di trasferimento gzip e deflate vengono automaticamente decodificate.

Ad esempio, per creare un'immagine da dati binari restituiti da una richiesta, è possibile utilizzare il seguente codice:

```
From PIL import Image
From io import BytesIO


i=Image.open(BytesIO(r.content))
```

Figura 4.15: Esempio creazione immagine

4.3.1.4. Contenuto della risposta JSON

Esiste anche un decoder JSON integrato, nel caso in cui si tratti di dati JSON:

```
import requests


r=request.get("https://api.github.com/events")
r.json()
# [{u "repository": {u "open_issues": 0, u "URL": "https: //github.com /
...
```

Nel caso in cui la decodifica JSON non riesca, r.json() solleva un'eccezione.

Ad esempio, se la risposta ottiene un 204 (nessun contenuto) o se la risposta contiene JSON non valido, il tentativo di r.json() genera ValueError: nessun oggetto JSON potrebbe essere decodificato.

Va notato che il successo della chiamata a r.json() non indica il successo della risposta. Alcuni server potrebbero restituire un oggetto JSON in una risposta non riuscita (ad es. Dettagli di errore con HTTP 500). Tale JSON verrà decodificato e restituito.

Per verificare che una richiesta abbia esito positivo, è necessario utilizzare r.raise_for_status() o verificare r.status_code.

4.3.1.5. Contenuto di risposta non elaborato

Nel caso in cui si desidera ottenere la risposta del socket raw dal server, è possibile accedere a r.raw. Per farlo è necessario assicurarsi di impostare stream = **True** nella richiesta iniziale.

Una volta fatto, si può fare:

```
r=request.get("https://api.github.com/events", stream=True)


r.raw
# <urllib3.response.HTTPResponse object at 0x101194810>


r.raw.read(10)
# '\x1f\x8b\x08\x00\x00\x00\x00\x00\x00\x03'
```

Figura 4.16: Esempio di codice per la risposta del socket raw

In generale, tuttavia, è necessario utilizzare un modello come quello in figura 4.17 per salvare ciò che viene trasmesso in streaming su un file:

```
with open(filename, "wb") as fd:
   for chunk in r.iter_content(chunk_size=128):
       fd.write(chunk)
```

Figura 4.17: Esempio di salvataggio di streaming

L'uso di Response.iter_content gestirà molto di ciò che altrimenti dovrebbe essere gestito dall'utente quando si utilizza direttamente Response.raw.

Durante lo streaming di un download, il modo in fig. 4.17 è il preferito e il consigliato per recuperare il contenuto. È da notare che chunk_size può essere regolato liberamente per adattarlo alle proprie esigenze.

Nota

Una nota importante sull'uso di Response.iter_content contro Response.raw.
Response.iter_content decodificherà automaticamente gzip e ridurrà le codifiche di trasferimento.
Response.raw è un flusso non elaborato di byte, non trasforma il contenuto della risposta.

Se vi è necessità di accedere ai byte così come sono stati restituiti, è necessario utilizzare Response.raw.

4.3.1.6. Intestazioni personalizzate

Se si desidera aggiungere intestazioni HTTP a una richiesta, si passa semplicemente un dict al parametro headers.

Ad esempio, non è stato specificato l'user-agent nell'esempio precedente:

```
url= "https://api.github.com/some/endpoint"
headers = {"user-agent" : "my-app / 0.0.1"}


r=request.get(url, headers=headers)
```

Figura 4.18: Esempio di specifica di user-agent

Nota

Le intestazioni personalizzate hanno meno priorità rispetto a fonti di informazione più specifiche. Ad esempio:

- Le intestazioni di autorizzazione impostate con le intestazioni = verranno sostituite se le credenziali sono specificate in .netrc, che a sua volta verrà sovrascritto dal parametro auth =;
- Le intestazioni di autorizzazione verranno rimosse se si viene reindirizzati off-host.
  Le intestazioni di autorizzazione proxy verranno sostituite dalle credenziali del proxy fornite nell'URL;
- Le intestazioni di Content-Length verranno ignorate quando si può determinare la lunghezza del contenuto.

Inoltre, Requests non cambia affatto il suo comportamento in base alle intestazioni personalizzate specificate. Le intestazioni vengono semplicemente passate nella richiesta finale.

Nota

Tutti i valori di intestazione devono essere una stringa, bytestring o unicode. Sebbene consentito, si consiglia di evitare di passare valori di intestazione unicode.

4.3.1.7. Richieste POST più complicate

Generalmente, si desidera inviare alcuni dati con codifica del modulo, proprio come un modulo HTML.

Per fare ciò, è sufficiente passare un dizionario all'argomento dati. Il dizionario dei dati verrà automaticamente codificato in forma al momento della richiesta:

```
payload={"key1" : "value1", "key2" : "value2"}


r=request.post("https://httpbin.org/post", data=payload)
print(r.text)
# {
#   "from": {
#         "key2": "value2",
#          "key1":"value1"
#   },
#  ...
# }
```

Figura 4.19: Esempio di passaggio di dizionario dati [1]

L'argomento dati può anche avere più valori per ogni chiave. Questo può essere fatto trasformando i dati in un elenco di tuple o in un dizionario con elenchi come valori.

Ciò è particolarmente utile quando il modulo ha più elementi che usano la stessa chiave:

```
payload_tuples = [("key1", "value1"), ("key1", "value2")]
r1 = requests.post("https://httpbin.org/post", data=payload_tuples)
payload_dict = {"key1": ["value1", "value2"]}
r2 = requests.post("https://httpbin.org/post", data=payload_dict)
print(r1.text)


r=request.post("https://httpbin.org/post", data=payload)
# {
#   “from”: {
#         “key1”:[
#             “value1”,
#            “value2”
#          ]
#   },
# }
r1.text == r2.text
# True
```

Figura 4.20: Esempio di passaggio di dizionario dati [2]

È anche possibile inviare dati che non sono codificati in forma. Se si passa una stringa anziché un dict, tali dati verranno pubblicati direttamente.

```
import requests


url="https://api.github.com/some/endpoint"
payload={"some":"data"}


r=requests.post(url, data=json.dumps(payload))
```

Figura 4.21: Esempio API GitHub v3 accetta i dati POST/PATCH con codifica JSON [1]

Invece di codificare tu stesso il dict, è possibile passarlo direttamente usando il parametro json (aggiunto nella versione 2.4.2) e verrà codificato automaticamente:

```
url="https://api.github.com/some/endpoint"
payload={"some": "data"}


r=requests.post(url, json.dumps(payload))
```

Figura 4.22: Esempio API GitHub v3 accetta i dati POST/PATCH con codifica JSON [2]

Nota

Il parametro json viene ignorato se vengono passati dati o file.

L'uso del parametro json nella richiesta cambierà il Content-Type nell'intestazione in application/json.

4.3.1.8. Codici di stato della risposta

È possibile controllare il codice di stato della risposta:

```
r=requests.get("https://httpbin.org/get")
r.status_code
# 200
```

Le richieste vengono inoltre fornite con un oggetto di ricerca del codice di stato incorporato per un facile riferimento:

```
r.status_code == requests.codes.ok
# True
```

Se è stata inviata una richiesta non valida (un errore client 4XX o una risposta di errore del server 5XX), è possibile sollevarla con Response.raise_for_status():

```
bad_r = requests.get("https://httpbin.org/status/404")
bad_r.status_code
# 404

bad_r.raise_for_status()

Traceback (most recent call last):
  File "requests/models.py", line 832, in raise_for_status
    raise http_error
requests.exceptions.HTTPError: 404 Client Error
```

Figura 4.23: Esempio di un possibile errore

Ma poiché lo status_code per r era 200, nell'esempio sopratiportato,quando viene chiamato raise_for_status() si ottiene:

```
r.raise_for_status
# None
```

4.3.1.9. Intestazioni di risposta

È possibile visualizzare le intestazioni di risposta del server usando un dizionario Python:

```
r.headers
# {
#     "content-encoding" : "gzip",
#     "transfer-encoding" : "chunked",
#     "connection" : "close",
#     "server" : "nginx/1.0.4",
#     "x-runtime" : "148ms",
#     "etag" : "e1ca502697e5c9317743dc078f67693f",
#     "content-type" : "application/json"
# }
```

Il dizionario è "speciale" ed è fatto unicamente per le intestazioni HTTP.
Secondo RFC 7230, i nomi delle intestazioni HTTP non fanno distinzione tra maiuscole e minuscole.

Quindi, è possibile accedere alle intestazioni usando qualsiasi maiuscola:

```
r.headers["Content-Type"]
# "application/json"

r.headers.get("content-type")
# "application/json"
```

È anche speciale in quanto il server avrebbe potuto inviare più volte la stessa intestazione con valori diversi, ma le richieste vengono combinate in modo che possano essere rappresentate nel dizionario all'interno di una singola mappatura, come da **RFC 7230**.

Un destinatario potrebbe combinare più campi di intestazione con lo stesso nome di campo in una coppia "nome-campo: campo-valore", senza modificare la semantica del messaggio, aggiungendo ciascun valore di campo successivo al valore di campo combinato in ordine, separati da un virgola.

4.3.1.10. Cookies

Se una risposta contiene alcuni cookie, è possibile accedervi rapidamente:

```
url= "https://example.com/some/cookie/setting/url"
r = requests.get(url)

r.cookies["example_cookie_name"]
# "example_cookie_value"
```

Figura 4.24: Accesso ai cookie

Per inviare i propri cookie al server, è possibile utilizzare il parametro cookies:

```
url= "https://httpbin.org/cookies"
cookies = dict(cookies_are = "working")

r=requests.get(url, cookies=cookies)
r.text
# {"cookies" : {"cookies_are" : "working"}}
```

Figura 4.25: Invio di cookie personalizzati

I cookie vengono restituiti in un RequestsCookieJar, che agisce come un dict ma offre anche un'interfaccia più completa, adatta per l'uso su più domini o percorsi.

I barattoli di cookie (Cookie Jar) possono anche essere passati a richieste:

```
jar = requests.cookies.RequestsCookieJar()
jar.set("tasty_cookie", "yum", domain= "httpbin.org", path= "/cookies")
jar.set("gross_cookie", "blech", domain= "httpbin.org", path=
"/elsewhere")


url = "https://httpbin.org/cookies"
r = requests.get(url, cookies=jar)
r.text
# {"cookies" : {"tasty_cookie" : "yum"}}
```

Figura 4.26: Esempio di Cookie Jar

4.3.1.11. Reindirizzamento e cronologia

Per impostazione predefinita, Requests eseguirà il reindirizzamento della posizione per tutti i verbi tranne HEAD.

È possibile utilizzare la proprietà history dell'oggetto Response per tenere traccia del reindirizzamento.

L'elenco Response.history contiene gli oggetti Response creati per completare la richiesta ed è ordinato dalla risposta più vecchia a quella più recente.

```
r=requests.get("http://github.com/")


r.url
# "https://github.com/"


r.status_code
# 200


r.history
# [<Response [301]>]
```

Figura 4.27: Esempio reindirizzamento di GitHub di tutte le richieste HTTP su HTTPS

Se si sta utilizzando GET, OPTIONS, POST, PUT, PATCH o DELETE, è possibile disabilitare la gestione del reindirizzamento con il parametro allow_redirects:

```
r=requests.get("http://github.com/", allow_redirects=False)


r.status_code
# 301


r.history
# []
```

Se stai usando HEAD, puoi anche abilitare il reindirizzamento:

```
r=requests.head("http://github.com/", allow_redirects=True)


r.url
# "https://github.com/"


r.history
# [<Response [301]>]
```

4.3.1.12. Timeouts

È possibile indicare alle Richieste di interrompere l'attesa di una risposta dopo un determinato numero di secondi con il parametro di timeout.

Quasi tutto il codice di produzione dovrebbe utilizzare questo parametro in quasi tutte le richieste. In caso contrario, il programma potrebbe bloccarsi indefinitamente:

```
requests.get("http://github.com/", timeout=0.001)


Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
requests.exceptions.HTTPError: HTTPConnectionPool(host='github.com',
port=80): Request timed out. (timeout=0.001)
```

Nota: il timeout non è un limite di tempo per il download dell'intera risposta; piuttosto, viene sollevata un'eccezione se il server non ha emesso una risposta per secondi di timeout (più precisamente, se non sono stati ricevuti byte sul socket sottostante per secondi di timeout).
Se non viene specificato esplicitamente alcun timeout, le richieste non scadono.

4.3.1.13. Errori ed eccezioni

In caso di un problema di rete (ad es. Errore DNS, connessione rifiutata, ecc.), Le richieste genereranno un'eccezione ConnectionError.

Response.raise_for_status() genererà un HTTPError se la richiesta HTTP ha restituito un codice di stato non riuscito.

Se una richiesta scade, viene sollevata un'eccezione di timeout. Se una richiesta supera il numero configurato di reindirizzamenti massimi, viene sollevata un'eccezione TooManyRedirects. Tutte le eccezioni che Requests genera esplicitamente le eredità da request.exceptions.RequestException.

4.3.2. Time

Python ha un modulo chiamato time per gestire le attività legate al tempo. Per utilizzare le funzioni definite nel modulo, dobbiamo prima importare il modulo. Ecco come:

```
import time
```

4.3.2.1. Funzioni

TIME.TIME

La funzione time() restituisce il numero di secondi trascorsi dall'epoca.

Per il sistema Unix, 1 gennaio 1970, 00:00:00 a UTC è epoca (il punto in cui inizia il tempo).

```
import time
seconds = time.time()
print("Seconds since epoch =", seconds)
```

TIME.CTIME

La funzione time.ctime() impiega i secondi trascorsi dall'epoca come argomento e restituisce una stringa che rappresenta l'ora locale.

```
import time

# seconds passed since epoch
seconds = 1545925769.9618232
local_time = time.ctime(seconds)
print("Local time: ", local_time)
```

Se esegui il programma, l'output sarà simile a:

```
Local time: Gio Dic 27 15:49:29 2019
```

TIME.SLEEP

La funzione sleep() sospende (ritarda) l'esecuzione del thread corrente per il dato numero di secondi.

```
import time

print("This is printed immediately")
time.sleep(2.4)
print("This is printed after 2.4 second")
```

TIME.LOCALTIME

La funzione localtime() impiega il numero di secondi trascorsi dall'epoca come argomento e restituisce struct_time nell'ora locale.

```
import time

result = time.localtime(1545925769)
print("Result: ", result)
print("\nYear: ", result.tm_year)
print("tm_hour: ", result.tm_hour)
```

Quando esegui il programma, l'output sarà simile a:

```
Result: time.struct_time(tm_year=2019, tm_mon=12, tm_mday=27,
tm_hour=15, tm_min=49, tm_sec=29, tm_wday=3, tm_yday=361,
tm_isdst=0)
Year: 2019
tm_hour: 15
```

Se nessun argomento o Nessuno viene passato a localtime(), viene utilizzato il valore restituito da time ().

TIME.GMTIME

La funzione gmtime() impiega il numero di secondi trascorsi dall'epoca come argomento e restituisce struct_time in UTC.

```
import time


result = time.gmtime(1545925769)
print("Result: ", result)
print("\nYear: ", result.tm_year)
print("tm_hour: ", result.tm_hour)
```

Quando esegui il programma, l'output sarà:

```
Result: time.struct_time(tm_year=2018, tm_mon=12, tm_mday=28,
tm_hour=8, tm_min=44, tm_sec=4, tm_wday=4, tm_yday=362, tm_isdst=0)
Year: 2019
tm_hour: 8
```

Se nessun argomento o *None* viene passato a gmtime(), viene utilizzato il valore restituito da time().

TIME.MKTIME

La funzione mktime() accetta struct_time (o una tupla contenente 9 elementi corrispondenti a struct_time) come argomento e restituisce i secondi trascorsi dall'epoca in ora locale. Fondamentalmente, è la funzione inversa di localtime().

```
import time


t = (2019, 12, 28, 8, 44, 4, 4, 362, 0)


local_time = time.mktime(t)
print("Local time: ", local_time)
```

L'esempio che segue mostra come mktime() e localtime() sono correlati.

```
import time


seconds = 1545925769
```

```
# returns struct_time
t = time.localtime(seconds)
print("T1: ", t)


# returns seconds from struct_time
s = time.mktime(t)
print("\s: ", seconds)
```

Quando esegui il programma, l'output sarà simile a:

```
T1:  time.struct_time(tm_year=2019, tm_mon=12, tm_mday=27, tm_hour=15,
tm_min=49, tm_sec=29, tm_wday=3, tm_yday=361, tm_isdst=0)
s: 1545925769.0
```

TIME.ASCTIME

La funzione asctime () accetta struct_time (o una tupla contenente nove elementi corrispondenti a struct_time) come argomento e restituisce una stringa che lo rappresenta. Ecco un esempio:

```
import time


t = (2019, 12, 28, 8, 44, 4, 4, 362, 0)


result = time.asctime(t)
print("Result: ", result)
```

Quando esegui il programma, l'output sarà:

```
Result: Ven Dic 28 08:44:04 2019
```

TIME.STRFTIME

La funzione strftime() accetta struct_time (o tupla corrispondente) come argomento e restituisce una stringa che lo rappresenta in base al codice di formato utilizzato. Per esempio

```
import time

named_tuple = time.localtime() # get struct_time
time_string = time.strftime("%m/%d/%Y, %H:%M:%S", named_tuple)

print(time_string)
```

Quando esegui il programma, l'output sarà simile a:

```
28/12/2019, 09:47:41
```

Qui, %Y, %m, %d, %H ecc. Sono codici di formato.

- **%Y :** anno [0001, ..., 2018, 2019, ..., 9999]
- **%m :** mese [01, 02, ..., 11, 12]
- **%d :** giorno [01, 02, ..., 30, 31]
- **%H :** ora [00, 01, ..., 22, 23
- **%M :** mese [00, 01, ..., 58, 59]
- **%S :** secondo [00, 01, ..., 58, 61]

TIME.STRPTIME

La funzione strptime() analizza una stringa che rappresenta time e restituisce struct_time.

```
import time


time_string = "21 Giugno, 2019"
result = time.strptime(time_string, "%d %B, %Y")


print(result)
```

Quando esegui il programma, l'output sarà:

```
time.struct_time(tm_year=2019, tm_mon=6, tm_mday=21, tm_hour=0,
tm_min=0, tm_sec=0, tm_wday=3, tm_yday=172, tm_isdst=-1)
```

4.3.2.2. Classi

TIME.STRUCT_TIME

Diverse funzioni nel modulo time come gmtime (), asctime () ecc. Prendono l'oggetto time.struct_time come argomento o lo restituiscono.

Ecco un esempio di oggetto time.struct_time.

```
time.struct_time(tm_year=2019, tm_mon=12, tm_mday=27,
                 tm_hour=6, tm_min=35, tm_sec=17,
                 tm_wday=3, tm_yday=361, tm_isdst=0)
```

Indice	Attributi	Valori
O	tm_year	0000, …, 2019, …, 9999
1	tm_mon	1, 2, …, 12
2	tm_mday	1, 2, …, 31
3	tm_hour	0, 1, …, 23
4	tm_min	0, 1, …, 59
5	tm_sec	0, 1, …, 61
6	tm_wday	0, 1, …, 6; Lunedì è 0
7	tm_yday	1, 2, …, 366
8	tm_isdst	0,1 o -1

I valori (elementi) dell'oggetto time.struct_time sono accessibili utilizzando sia gli indici che gli attributi.

CAPITOLO 5:
PROGETTO

Dovo aver discusso di tutti i fondamenti necessari per la comprensione del progetto in questo capitolo si discuterà del progetto stesso, del perché è stato necessario creare un codice e si mostrerà una visione generale del progetto concluso.

Il progetto in sé consiste nella connessione, tramite protocollo MQTT, di sensori già installati con una piattaforma selezionata per l'acquisizione dei dati e la monitorizzazione dell'ambiente.

5.1. Piattaforma

Come già detto in precedenza la piattaforma scelta per il progetto è la Ubidots for Education, poiché propone molteplici strumenti ad un prezzo praticamente nullo.

Per effettuare una connessione non-TLS con la piattaforma sono necessarie le credenziali del proprio account di Ubidots (TOKEN e Password) e va effettuata come di norma tramite la porta 1883.

5.1.1. Il problema

Avendo scelto la versione Education, per la connessione non è possibile utilizzare un endpoint personale ma è d'uopo utilizzare l'indirizzo broker predefinito "things.ubidots.com".
Inoltre, la piattaforma accetta come ingresso dati solo pacchetti JSON (*JavaScript Object Notation*).

Il codice presente sui sensori già installati, però, invia i dati tramite protocollo MQTT ad un broker personalizzato ossia "193.205.130.248".
Dunque, vi è un fondamentale problema di comunicazione tra la piattaforma e ciò che è già stato installato.

5.1.2. La soluzione

La soluzione più semplice sarebbe stata quella fare un upgrade dalla versione Education alla versione integrale per le aziende. Ma questa scelta sarebbe stata a dir poco troppo costosa.
Dopo aver parlato con il servizio di marketing della piattaforma si è riscontrato che l'unica soluzione proposta per l'utilizzo di un indirizzo broker personalizzato era in un piano tariffario dal costo di partenza piuttosto elevato.

Un'altra soluzione per ovviare a questo problema di comunicazione era di scrivere un codice da implementare su di una scheda Raspberry.
Questo codice avrebbe dovuto iscriversi al broker personalizzato a cui i sensori inviavano i pacchetti di dati, scomporre questi pacchetti e creare dei pacchetti JSON da inoltrare all'indirizzo privato della piattaforma.

Questo codice è stato scritto in Python3 in tre diverse versioni per testare il funzionamento del programma.
La prima versione è una semplice iscrizione all'indirizzo "193.205.130.248" per ricevere i pacchetti e una volta capito come arrivavano i pacchetti questo esperimento è stato utile per scomporli e avere i diversi dati a disposizione come delle semplici variabili.
La seconda versione un po' più complessa prevedeva la creazione di pacchetti JSON impostati come richiesto dalla piattaforma ma come payload vi erano dati generati casualmente per testare l'efficacia di tale codice e l'eventuale ricezione o perdita di dati della piattaforma.
La terza ed ultima versione è stata un'unione delle due versioni precedenti con alcune linee di codice per il debug istantaneo da parte dell'utente.

Per rendere lo scritto più fluido e meno ripetitivo possibile nella sezione seguente sarà riportato solo il codice finale.

5.2. Codice

Prima di esaminare il codice è necessario prima capire perché Python viene utilizzato al posto di altri linguaggi, come C ++.
Mentre altre lingue forniscono velocità, Python è senza dubbio il linguaggio di proto-tipizzazione finale grazie al suo sistema di libreria di facile utilizzo con PIP.
Ad esempio, far funzionare un client MQTT in C ++ potrebbe comportare alcune sfide poiché alcune librerie funzionano solo su sistemi Windows e altre hanno complesse opzioni di installazione e collegamento.
Python necessita dell'utilizzo di un comando PIP per installare la libreria. Da lì, tutti i programmi Python possono importare la libreria MQTT.
Inoltre, la libreria MQTT per Python è incredibilmente facile da usare e richiede solo una manciata di righe di codice.

Il primo passo per usare MQTT con Python è installare la libreria Phao-MQTT. Per fare ciò su Raspberry Pi, è necessario aprire una finestra della console e inserire il comando seguente.

```
#In order to make this program work you need to install the paho client.
#To install paho-mqtt library, use the following commands from command
prompt:
#Python 3.x users can use pip3 command --> pip3 install paho-mqtt
#Python 2.x users can use pip2 command --> pip2 install paho-mqtt
```

Questa è un'istruzione di installazione PIP che troverà, scaricherà e installerà automaticamente la libreria Phao-MQTT. Ed è anche la prima linea di commento all'interno del codice per renderlo in più leggibile possibile.

```
#Import useful libraries
import paho.mqtt.client as mqtt #Paho-client
import time
import requests
import math
```

Qui vengono importate e incluse all'interno del programma le librerie spiegate nel capitolo precedente.

```
#Define global variables
TOKEN = "A1E-606L25zbFIPyy4xjPkKBe6N7b1eJaE"  #Ubidots TOKEN/User
DEVICE_LABEL = "sensor_"
VARIABLE_LABEL_1 = "temperature"
VARIABLE_LABEL_2 = "humidity"
VARIABLE_LABEL_3 = "voc"
value_1 = 0 #Id sensor
value_2 = 0 #VOC
value_3 = 0 #Humidity
value_4 = 0 #Temperature
```

La definizione delle variabili globali è necessaria per il richiamo e l'utilizzo di queste all'interno di diverse funzioni.

Come commentato all'interno del codice stesso il TOKEN qui definito è unico per ogni account personale di Ubidots, quindi nel caso di un diverso utilizzo del codice è necessario sostituirlo con il proprio TOKEN.

```
#Build the payload with the variables from on_message
def build_payload(variable_1, variable_2, variable_3):
    #Import the global values for sending data
    global value_2
    global value_3
    global value_4
    #Build the json payload
```

```
    payload = {variable_1: value_4, variable_2: value_3, variable_3:
value_2}

    return payload
```

Qui è stata definita una funzione per la creazione del payload del pacchetto JSON. Utilizzando le variabili globali value_2, value_3, value_4 sopra definite e di seguito assegnate rispettivamente come Voc, Umidità e Temperatura.

```
#Create and send the json message

def post_request(payload):

    #Import the global value for the sensor identification

    global value_1

    #Create the headers for the HTTP requests

    url = "http://things.ubidots.com"

    url = "{}/api/v1.6/devices/{}".format(url,
DEVICE_LABEL+str(value_1))

    headers = {"X-Auth-Token": TOKEN, "Content-Type":
"application/json"}


    #Make the HTTP requests

    status = 400

    attempts = 0

    while status >= 400 and attempts <= 5:

        req = requests.post(url=url, headers=headers, json=payload)

        status = req.status_code

        #Increment the attempts (max. 5) and wait 1s before retry

        attempts += 1

        time.sleep(0.1)


    #Processes results

    if status >= 400:

        print("[ERROR] Could not send data after 5 attempts, please
check your token credentials and internet connection")

        return False


    print("[INFO] Request made properly, your device is updated")

    return True
```

Successivamente è stata implementata la funzione post_request come definito nelle librerie precedenti.

Questa funzione è necessaria per la creazione di una richiesta HTTP per eseguire dei tentativi ripetuti di invio del pacchetto JSON.

Vi sono delle righe di codice addizionali per un riscontro immediato del successo o del fallimento dell'invio del pacchetto con delle stampe dunque inutili per il funzionamento del codice ma utili per la revisione dello stesso.

```
#Start the republishing process
def publish():
    payload = build_payload(VARIABLE_LABEL_1, VARIABLE_LABEL_2,
VARIABLE_LABEL_3)
    print("[INFO] Attemping to send data")
    post_request(payload)
    print("[INFO] Finished")
```

La funzione sopra riportata richiama la costruzione del payload del pacchetto e la richiesta di invio.

Necessaria quindi per inoltrare i pacchetti ricevuti sul broker personalizzato questa funzione viene richiamata ogni volta che si riceve un pacchetto, in un loop infinito.

```
#Callbacks functions from paho-mqtt library
#Check the log data
def on_log (client, userdata, level, buf):
        print("Log: "+buf)
```

Questa funzione non viene poi richiamata poiché è completamente inutile per l'esecuzione del programma ma potrebbe essere utilizzata per sessioni di debug come visto nella sua spiegazione nel capitolo relativo alle librerie di Python.

```
#Connection to the broker with a proper result code (rc)
def on_connect ( client, userdata, flags, rc):
        if rc == 0:
                print("[INFO] Connected with code: " +str(rc))
        else:
                print("[ERROR] Bad connection returned code: ", rc)
```

Alla connessione con il broker segue una stampa utile solo per un riscontro visivo immediato sullo stato della connessione, dunque non essenziale ai fini dell'esecuzione del codice.

```
#Recive a message, decode the message and start the resend() function
def on_message (client, userdata, msg):
        #Assign the message topic to the variable topic in order to
check if it is the right one
        topic = msg.topic
        #Assign the message payload after decoding and transformed into
a string to the variable m_decode
        m_decode = str(msg.payload.decode("utf-8"))
        print ("[INFO] Message received :", m_decode)
        #Unpack the variable m_decode into single chars
        a,b,c,d,e,f,g,h,i,l,m,n,o,p,q,r,s,t,u = m_decode
        #Import the global variables values in order to assign it to a
part of the message
        global value_1
        value_1 = int(a)
        global value_2
        value_2 = float(c+d+e+f+g+h)
        global value_3
        value_3 = float(l+m+n+o+p)
        global value_4
        value_4 = float(r+s+t+u)
        #Print the variables in order to check if it is the right output
        print("Id sensore :", value_1)
        print("Voc :", value_2)
        print("Umidità :", value_3,"%")
        print("Temperatura :", value_4,"°C")
        #Republish the message
        publish()
```

Quella soprariportata è la funzione fondamentale per la ricezione del messaggio e la sua decodifica.

Una volta ricevuto il messaggio viene decodificato e scomposto cifra per cifra per poi venire nuovamente assemblato all'interno di diverse variabili le quali come detto in precedenza vengono utilizzate per comporre il carico del messaggio JSON. Richiama in conclusione la funzione per l'inoltro del pacchetto.

Anche all'interno di questa funzione vi sono presenti linee utili unicamente ad un riscontro visivo immediato per una veloce assicurazione del funzionamento del codice.

```
#Create a new instance

client = mqtt.Client()

#client.on_log = on_log --> Used only in order to verify the log
credentials

client.on_connect = on_connect #Attach funciotn to callback

client.on_message = on_message #Attach funciotn to callback


#Connect the client to the broker:port

client.connect("193.205.130.248", 1883, 60)
#client.connect("broker/host", port, keepalive)

#client.username_pw_set("Username", "Password") --> Useless in our
broker

#Subscribe to the topic

client.subscribe("sensor_readings/#")


#Start a neverending loop

client.loop_forever()
```

Le ultime righe sono state utilizzate per il richiamo delle funzioni e per il loro effettivo funzionamento, il quale è garantito grazie alla funzione client.loop_forever().

5.3. Conclusioni

Non resta che utilizzare propriamente la piattaforma per gestire e controllare i dati ricevuti dai vari sensori.
All'avvio della piattaforma si ha la possibilità di creare una nuova Dashboard (come mostrato in Figura 5.1), ovvero una pagina in cui inserire grafici e comandi per ognuno dei sensori.

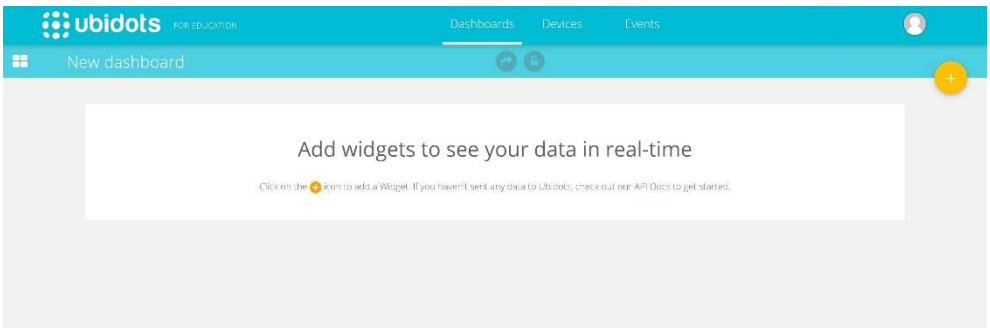


Figura 5.1: Schermata di creazione Dashboard - Ubidots

Una volta creata la propria Dashboard è necessario creare dei sensori virtuali (Figura 5.2) nei quali verranno raccolti ed immagazzinati i dati ricavati dai sensori reali. Per semplicità di comprensione

è utile far notare che nel nostro caso i sensori sono stati chiamati S.2, S.3, S.4 poiché avrebbero dovuto essere 4 ma al momento della scrittura di questo testo il sensore S.1 non era operativo.

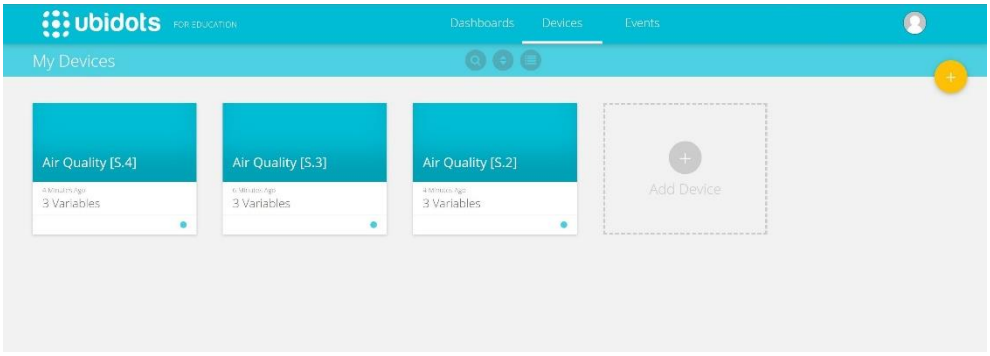


Figura 5.2: Schermata di creazione Devices - Ubidots

Una volta creati i sensori virtuali, per ognuno di essi è necessario creare delle variabili, ognuna delle quali registrerà i dati che arrivano dai sensori fisici. Nel nostro caso saranno le variabili di temperatura, umidità e VOC come mostrato in Figura 5.3. Si può anche aggiungere la geolocalizzazione ad ognuno dei sensori.

Per non dilungarsi troppo con immagini ripetitive verrà riportato l'esempio di un solo sensore.

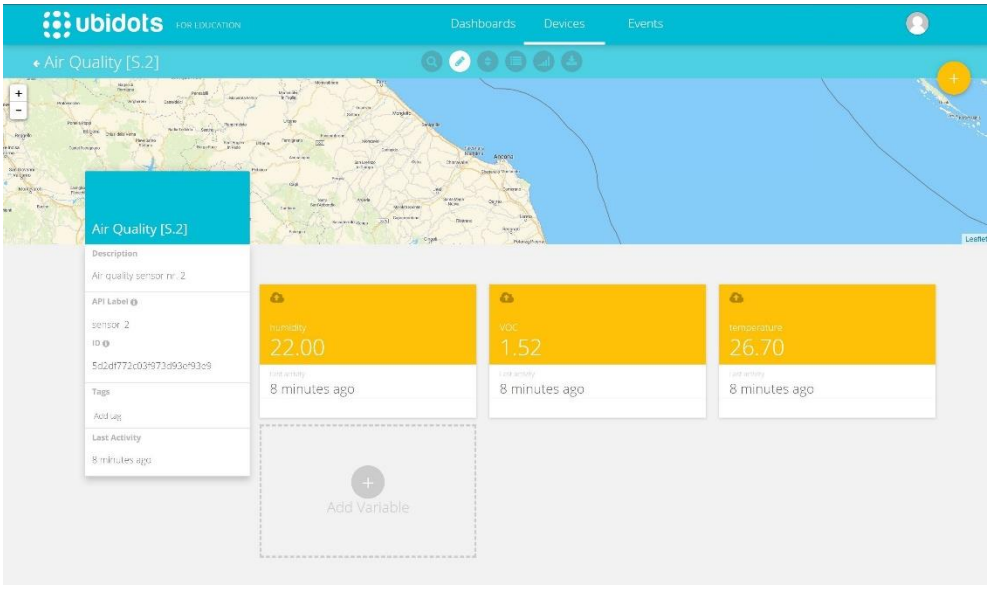


Figura 5.3: Schermata del sensore - Ubidots

Com'è possibile notare le variabili mostrano non solo l'ultimo dato ricevuto ma anche il momento in cui è stato ricevuto e quindi l'ultima comunicazione.

I dati immagazzinati sono quindi a disposizione dell'utente e possono essere utilizzati ad esempio per creare grafici, indicatori, tabelle, ecc…. con i dati immagazzinati. Nel nostro caso vi erano tre sensori e si è preferito creare una Dashboard per ogni sensore per mettere in evidenza e in mostra:

- L'andamento della temperatura;
- La temperatura attuale;
- La temperatura media;

- La temperatura massima;
- L'andamento dell'umidità;
- L'umidità attuale;
- L'umidità media;
- L'andamento del livello di VOC;
- Il livello di VOC attuale;
- Uno storico di 72h dei dati ricevuti di tutte e tre le variabili (Uno per ogni variabile: Temperatura, umidità e VOC).

Nelle figure seguenti (5.4, 5.5, 5.6, 5.7) verranno mostrate le Dashboard di tutti e tre i sensori con gli elementi sopra elencati e una quarta Dashboard che mette a confronto i dati ricevuti da tutti e tre i sensori mettendo in evidenza:

- Gli andamenti delle temperature;
- Gli storici delle temperature;
- Gli andamenti delle umidità;
- Gli storici delle umidità;
- Le variazioni dei livelli di VOC;
- Gli storici dei livelli di VOC;
- Uno storico di tutte le variabili di tutti i sensori (per il controllo della perdita dei dati).

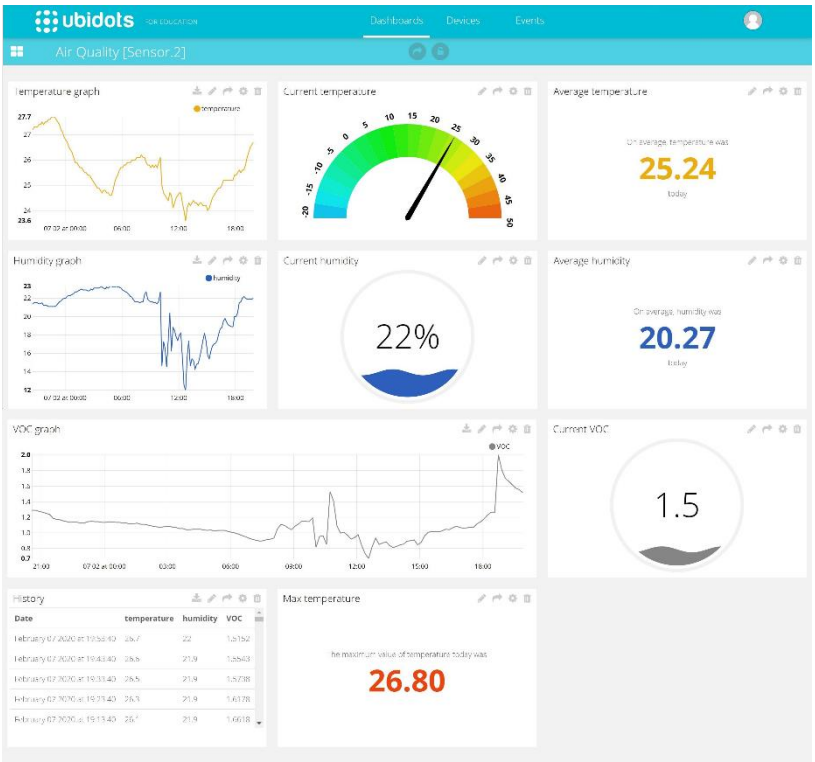


Figura 5.4: Dashboard relativo al sensore nr.2 - Ubidots

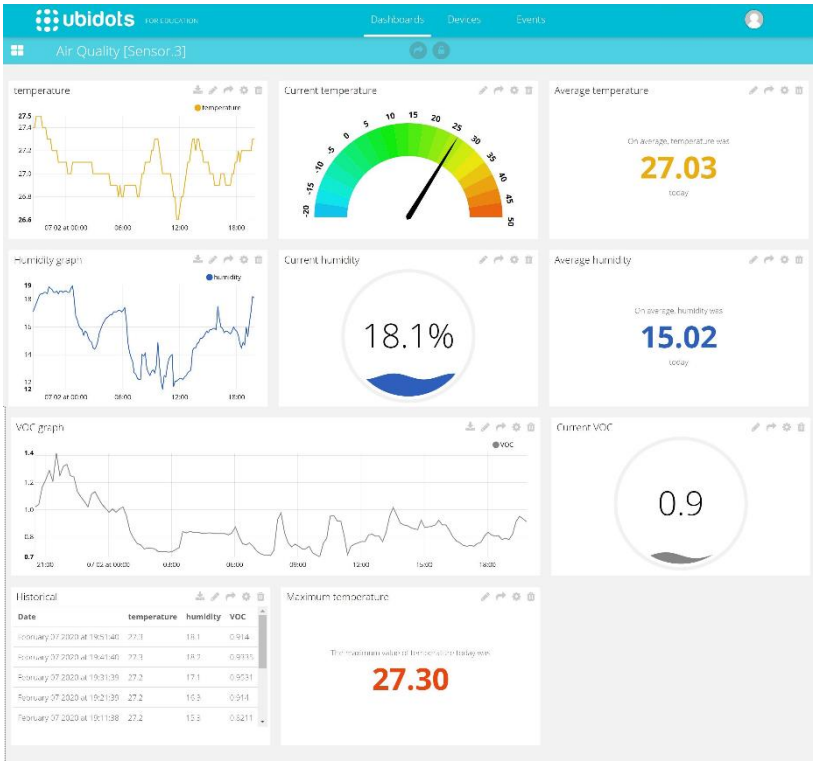


Figura 5.5: Dashboard relativo al sensore nr.3 - Ubidots

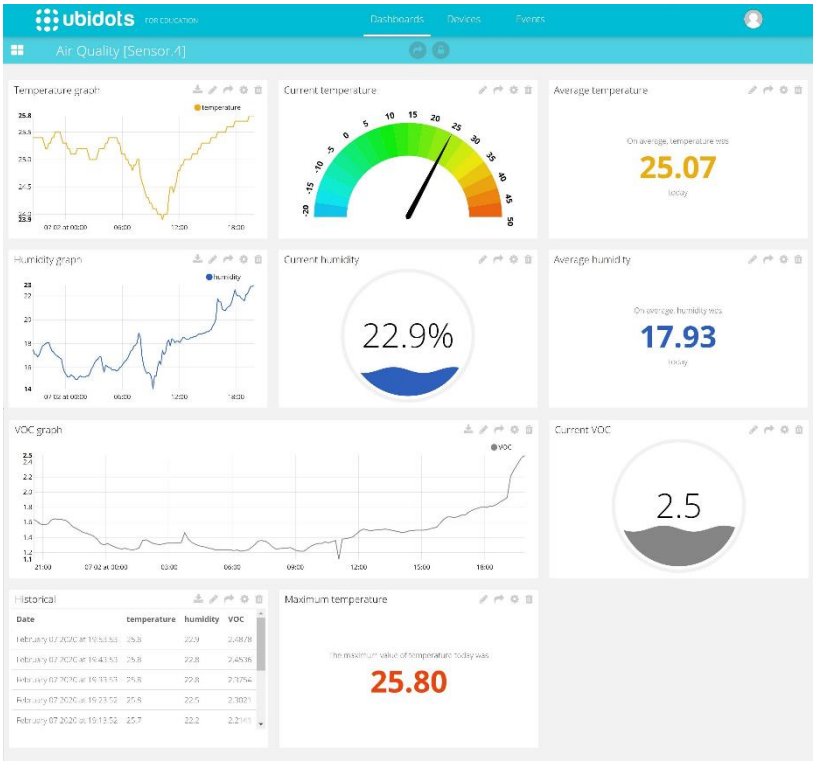


Figura 5.6: Dashboard relativo al sensore nr.4 - Ubidots

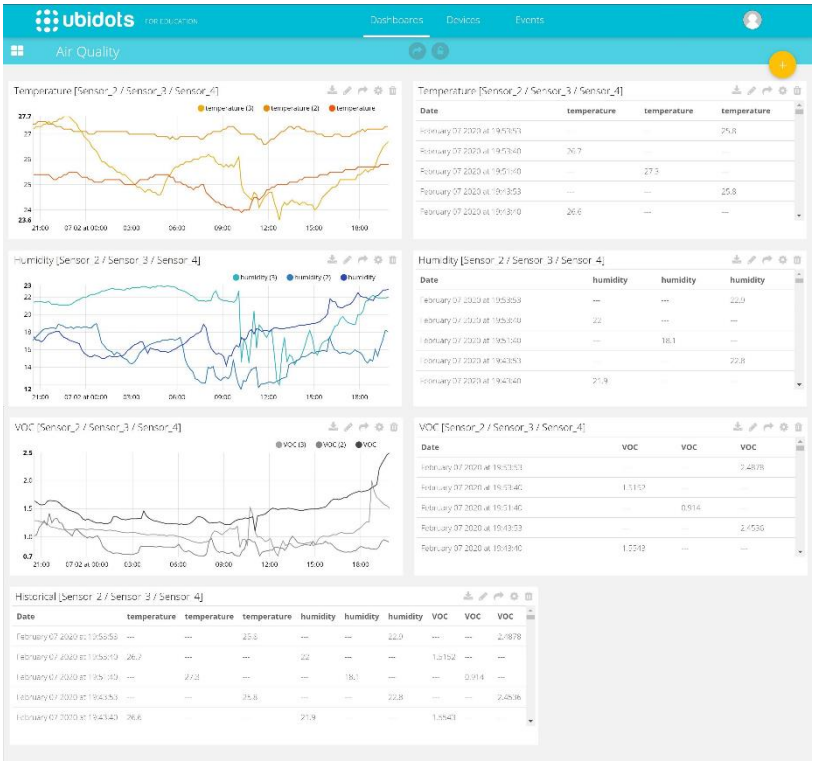


Figura 5.7: Dashboard relativo a tutti i sensori - Ubidots

La gestione delle Dashboard e dei sensori è interamente lasciata all'utente. Nell'ultima schermata, denominata Events, è possibile creare degli eventi, che consistono in azioni automatizzate e quindi definite a priori. Questi eventi vengono azionati dall'utente stesso o da una variabile e possono consistere nell'avvisare gli iscritti al broker, tramite mail o sms, modificare una determinata variabile e quindi eseguire un'azione automatizzata, avviare una "cascata" di eventi, ecc.…

Tramite la piattaforma è dunque possibile gestire, controllare e monitorare tutto quello che viene rilevato dai sensori. Uno strumento di semplice utilizzo ma molto efficace in cui è possibile creare un network IoT per qualsiasi campo di studio.

BIBLIOGRAFIA & SITOGRAFIA

- Manyika, James, Michael Chui, Peter Bisson, Jonathan Woetzel, Richard Dobbs, Jacques Bughin, and Dan Aharon. "*The Internet of Things: Mapping the Value Beyond the Hype*." McKinsey Global Institute, June 2015. p.3.
- https://en.wikipedia.org/wiki/IP_address
- http://www.treccani.it/enciclopedia/legge-di-moore_%28Enciclopedia-della-Scienza-e-della-Tecnica%29/
- https://www.mckinsey.com/mgi/overview/about-us
- https://en.wikipedia.org/wiki/Radio-frequency_identification
- https://www.camcode.com/asset-tags/what-are-rfid-tags/
- https://it.rs-online.com/web/generalDisplay.html?id=i/iot-internet-of-things
- Tschofenig, H., et. al., "*Architectural Considerations in Smart Object Networking*". Tech. no. RFC 7452. Internet Architecture Board, Mar. 2015.
- https://it.wikipedia.org/wiki/Trigger_(basi_di_dati)
- https://www.iab.org/2015/03/20/rfc-7452-architectural-considerations-in-smart-object-networking/
- https://it.wikipedia.org/wiki/Representational_State_Transfer
- https://en.wikipedia.org/wiki/Streaming_Text_Oriented_Messaging_Protocol
- https://en.wikipedia.org/wiki/SMS
- https://www.linkedin.com/pulse/internet-things-part-2-mahendra-bhatia
- https://www.researchgate.net/figure/Back-End-Data-Sharing-Model-10_fig1_301693195
- https://blog.teamleadercrm.it/product/silos-di-dati
- https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=14&ved=2ahUKEwjKq9bxpo3mAhXS2qQKHZa0BqsQFjANegQICRAB&url=https%3A%2F%2Fwww.webopedia.com%2FTERM%2FA%2FAPI.html&usg=AOvVaw2uqtLW-CJvbHI75qyvJNKu
- https://it.wikipedia.org/wiki/Gateway_(informatica)
- https://it.wikipedia.org/wiki/Marca_temporale
- https://it.wikipedia.org/wiki/Wi-Fi
- https://it.wikipedia.org/wiki/Bluetooth
- https://it.wikipedia.org/wiki/Ethernet
- https://www.thethingsnetwork.org/docs/lorawan/
- https://en.wikipedia.org/wiki/Sigfox
- https://internetofthingswiki.com/
- https://en.wikipedia.org/wiki/Eurotech_(company)
- https://en.wikipedia.org/wiki/IBM
- https://www.iso.org/standard/69466.html
- https://www.guru99.com/tcp-ip-model.html

- https://en.wikipedia.org/wiki/Internet_Assigned_Numbers_Authority
- https://www.digicert.com/ssl/
- https://www.mrwebmaster.it/primi-passi/backend_12722.html
- http://mqtt.org/
- https://en.wikipedia.org/wiki/MQTT
- https://subscription.packtpub.com/book/application_development/9781787287815/1/ch01lvl1sec18/understanding-wildcards
- https://www.cloudamqp.com/blog/2014-12-03-what-is-message-queuing.html
- http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html#_Toc398718035
- https://www.archerims.com/2015/08/31/the-practicality-of-being-data-agnostic/
- https://www.wifi-professionals.com/2019/01/4-way-handshake
- https://it.wikipedia.org/wiki/IEEE_802.15.4
- https://www.techopedia.com/definition/13460/user-datagram-protocol-udp
- C. Bormann Z. Shelby, K. Hartke. "*Constrained Application Protocol (CoAP)*" http://tools.ietf.org/search/draft-ietf-core-coap-18 , 2013.
- https://coap.technology/
- https://en.wikipedia.org/wiki/Constrained_Application_Protocol
- https://tools.ietf.org/html/rfc7275
- https://it.wikipedia.org/wiki/Throughput
- https://en.wikipedia.org/wiki/Internet_Engineering_Task_Force
- https://en.wikipedia.org/wiki/Proxy_server
- https://en.wikipedia.org/wiki/Protocol_converter
- http://www.treccani.it/enciclopedia/datagramma/
- https://www.bucap.it/focus/conservazione-digitale/conservazione-digitale-i-metadati
- https://tools.ietf.org/html/rfc6347
- https://it.wikipedia.org/wiki/Error-correcting_code
- http://www.pc-facile.com/glossario/ecc/
- https://it.wikipedia.org/wiki/Advanced_Encryption_Standard
- https://en.wikipedia.org/wiki/RSA_(cryptosystem)
- https://en.wikipedia.org/wiki/Multicast
- https://en.wikipedia.org/wiki/Network_address_translation
- https://en.wikipedia.org/wiki/Tunneling_protocol
- https://en.wikipedia.org/wiki/Port_forwarding
- https://www.eclipse.org/community/eclipse_newsletter/2014/february/article2.php
- https://www.zerounoweb.it/mobility/internet-of-things-serve-una-piattaforma-basata-sulledge-computing/
- https://www.kaaproject.org/what-is-iot-platform
- https://en.wikipedia.org/wiki/Edge_computing
- https://en.wikipedia.org/wiki/Provisioning_(telecommunications)
- https://en.wikipedia.org/wiki/Over-the-air_programming
- https://frontendmasters.com/books/front-end-handbook/2018/what-is-a-FD.html

- https://searchcloudcomputing.techtarget.com/definition/Platform-as-a-Service-PaaS
- https://www.kaaproject.org/
- https://cloud.oracle.com/OIC
- https://docs.oracle.com/en/cloud/paas/integration-cloud-service/gj-integration/what-is-integration-cloud.html
- https://aws.amazon.com/iot-core/?nc=sn&loc=2&dn=3
- https://cloud.google.com/solutions/iot/?hl=it
- https://azure.microsoft.com/it-it/overview/iot/
- https://docs.microsoft.com/en-us/azure/iot-fundamentals/iot-introduction
- https://docs.microsoft.com/en-us/azure/iot-fundamentals/iot-services-and-technologies
- https://www.altairsmartworks.com/try-for-free
- https://ubidots.com
- https://www.horsa.com/it/en/internet-of-things/horsa-iot-platform
- https://www.hpe.com/it/it/solutions/iot-platform.html
- http://www.stoorm5.com/piattaforma-iot/
- https://it.wikipedia.org/wiki/Batch
- https://en.wikipedia.org/wiki/WebSocket
- https://en.wikipedia.org/wiki/Software_development_kit
- https://it.wikipedia.org/wiki/Software_as_a_service
- https://searchitoperations.techtarget.com/definition/zero-touch-provisioning-ZTP
- https://pubs.vmware.com/identity-manager-26/index.jsp?topic=%2Fcom.vmware.wsp-administrator_26%2FGUID-C9070D4C-07EB-4970-8DC5-F657D0F47C90.html
- https://www.webopedia.com/TERM/OEM
- https://en.wikipedia.org/wiki/Onboarding
- https://it.wikipedia.org/wiki/Total_Cost_of_Ownership
- https://docs.python.org/3.8/tutorial/
- http://www.steves-internet-guide.com/into-mqtt-python-client/
- https://pypi.org/project/paho-mqtt/
- https://2.python-requests.org/en/master/user/quickstart/
- https://docs.python.org/3/library/time.html
- https://www.programiz.com/python-programming/time
- https://realpython.com/python-time-module/