

Università degli Studi di Genova

Facoltà di Ingegneria



Tesi di Laurea Magistrale in Ingegneria Elettronica

**PROGETTO E IMPLEMENTAZIONE DI UN
MOTORE DI ESECUZIONE
MULTIPIATTAFORMA PER APPLICAZIONI
IoT**

Relatore:

Chiar.mo Prof. Riccardo Berta

Candidati:

Luca Lazzaroni

Andrea Mazzara

24 Luglio 2020

A te, fra vent'anni.

Luca

Questa tesi è dedicata a Roberto,
che mi ha ispirato a diventare ingegnere.

Andrea

Sommario

Le applicazioni dell'Internet of Things (IoT) richiedono spesso una notevole larghezza di banda, bassa latenza e performance affidabili, e al tempo stesso devono rispettare requisiti normativi e di conformità, motivo per cui il Cloud Computing non risulta adatto in questi particolari casi applicativi.

Per ovviare ai problemi sopracitati, negli ultimi anni si sta affermando un nuovo approccio, l'Edge Computing: un'architettura distribuita di micro data center, ciascuno in grado di immagazzinare ed elaborare i dati a livello locale e, in seguito, trasmetterli ad un data center centralizzato o a un database su Cloud.

Edge Engine nasce allo scopo di realizzare un motore il più generico possibile e slegato dall'hardware, tale da raccogliere dati provenienti dai dispositivi ad esso collegati, elaborarli e inviarli su Cloud.

L'elaborato in questione tratta lo sviluppo di Edge Engine per dispositivi di tipo PC (Windows/ Linux/ MacOS). Il linguaggio di programmazione utilizzato è il C++, con l'intento di ottenere un prodotto finale multiplatforma, caratteristica concorde con i requisiti preposti di genericità e indipendenza dall'hardware.

Una volta completato lo sviluppo del sistema, ne verranno illustrate le potenzialità, dapprima in un contesto reale, successivamente in ambito virtuale. Nel primo caso si utilizzerà l'engine per il trattamento di dati provenienti da PC (info su RAM e ROM). Nel secondo invece, verrà creato un componente su Unity3D in grado di usufruire dei servizi offerti da Edge Engine, ma applicati a dati ottenuti dalla scena di gioco.

In ultimo, al fine di ottenere un resoconto riguardo l'effettiva efficacia del sistema, oltre che possibili spunti su eventuali criticità, verrà illustrato un esempio di utilizzo della libreria, in ambiente Arduino, da parte di due tesisti triennali.

Indice

1	Introduzione	1
2	Stato dell'Arte	6
3	Contesto di sviluppo	12
3.1	Measurify	13
3.2	Edge Engine	15
3.3	Evoluzione del progetto	17
4	Implementazione	19
4.1	Librerie POCO	19
4.1.1	Installazione	20
4.2	Utilizzo	23
4.3	Classi wrapper	25
4.3.1	Connection_windows	25
4.3.2	APIRest_windows	27
4.3.3	myDefines.h	29
4.4	Esempio di prova	30
4.5	Creazione della libreria	31
5	Esperimenti	34

5.1	Applicazione Windows	34
5.2	Applicazione alla Realtà Virtuale	38
5.2.1	Plugin	39
5.2.2	Unity	42
5.2.3	Applicazione reale	47
6	Conclusioni e Lavori Futuri	50

Elenco delle figure

1.1	Struttura dell'Edge Computing	2
1.2	Confronto tra Edge e Cloud Computing	3
1.3	Schema funzionale di Edge Engine per PC	5
2.1	Schema funzionale di Edge Engine su piattaforme eterogenee . .	11
3.1	Ecosistema IoT basato sull'Edge Computing	12
3.2	Struttura del server Cloud Measurify	14
3.3	Architettura Edge Engine - Measurify	17
3.4	Esempio di codice dipendente dalla piattaforma nella classe <code>Connection</code>	18
4.1	Confronto tra le due implementazioni del metodo <code>isConnected</code> .	27
4.2	Confronto tra le due implementazioni del metodo <code>PostLogin</code> . .	28
4.3	<code>myDefines.h</code>	29
4.4	Esempio di utilizzo della direttiva <code>#ifdef</code>	30
5.1	Dato relativo al valore di memoria RAM disponibile convertito in MB e conservato sul Cloud	37
5.2	Scena di gioco Unity3D	42
5.3	Variabili settabili dall'utente all'interno dell'inspector Unity . .	43
5.4	Metodi <code>Start</code> e <code>Update</code> dello script Unity	45

5.5	Dato relativo al numero di collisioni effettuate dalla pallina conservato sul Cloud	46
5.6	Schema funzionale di Edge Engine per Arduino	47
5.7	Dato relativo al rilevamento di presenza effettuato dal sensore PIR	48

Elenco delle tabelle

4.1	HTTP requests generalmente eseguite da Edge Engine	31
5.1	Parametri Measurify della risorsa Windows	35
5.2	Script correlati al device pc-probe	36
5.3	Parametri Measurify della risorsa Unity	43

Capitolo 1

Introduzione

Negli ultimi anni, la mole di dati prodotta da aziende e privati sta crescendo esponenzialmente: entro il 2022 si stima che, in media, si avranno 50 dispositivi connessi a Internet per abitazione [1]. Ovviamente, questi dati necessitano di essere processati e conservati, oltre che, all'occorrenza, condivisi tra più dispositivi. A tal fine, l'approccio che si è adottato maggiormente negli ultimi anni è quello del Cloud Computing: i dati non vengono processati in locale per mancanza di risorse, ma inviati a specifici data center online in grado di elaborarli e processarli, oltre che conservarli. Tale modalità introduce però alcune criticità:

- **Latenza:** in molti ambiti è richiesta un'elaborazione dei dati in tempo reale, si pensi per esempio ad un'eventuale applicazione che permetta ad un veicolo autonomo di riconoscere i pedoni. In questo specifico caso è richiesta una bassissima latenza dato l'enorme rischio in gioco. Tuttavia, proprio l'invio dei dati al Cloud, la successiva elaborazione degli stessi e, infine, l'invio di un feedback al dispositivo in uso, introducono ritardi non trascurabili. Pertanto, in questi specifici casi, il Cloud Computing risulta non essere l'approccio migliore.

- **Scalabilità:** l'invio dei dati al Cloud è problematico in tal senso, dato soprattutto il numero in crescita esponenziale di dispositivi connessi. Inoltre, l'invio di tutti i dati al Cloud è inefficiente in termini di consumo di risorse, specialmente se non è necessaria l'elaborazione dell'intero dataset.
- **Privacy:** l'invio di dati sensibili a server online aumenta i rischi di furto di tali informazioni, oltre al fatto che l'utente spesso e volentieri non è a conoscenza di come questi dati verranno trattati, né tantomeno di dove saranno conservati.

Una possibile soluzione a queste tre criticità, proprie del Cloud Computing, è l'Edge Computing (si veda figura 1.1).

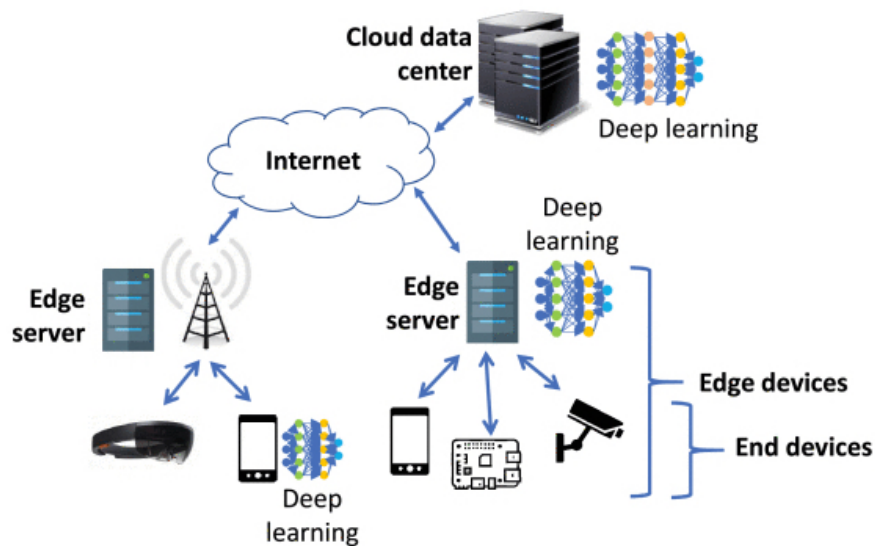


Figura 1.1: Struttura dell'Edge Computing

Tale approccio prevede una rete di micro data center posti nelle vicinanze dei dispositivi che rilevano i dati da elaborare. Proprio questa vicinanza alle

sorgenti dei dati permette di ridurre drasticamente la latenza (si veda figura 1.2). Inoltre, al fine di incrementare le prestazioni in termini di scalabilità, è prevista una struttura gerarchica dei dispositivi connessi, oltre al fatto che non risulta necessario apportare modifiche o espansioni ai data center in Cloud siccome i dati vengono elaborati in locale. Per quanto riguarda infine i vincoli di privacy, l'Edge Computing prevede l'elaborazione dei dati alla sorgente, solitamente grazie a un server locale, perciò non ne avviene la trasmissione sulla rete globale, riducendo dunque i rischi che ne deriverebbero.

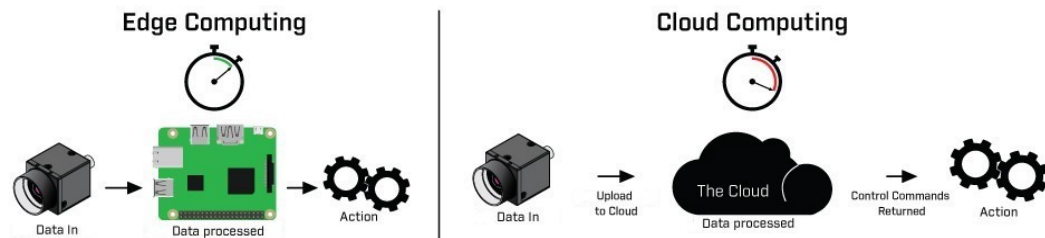


Figura 1.2: Confronto tra Edge e Cloud Computing

L'approccio Edge Computing presenta però alcune criticità.

Uno degli aspetti più importanti da considerare è l'elevata quantità di risorse richiesta da determinati algoritmi (come ad esempio quelli di Machine Learning o di elaborazione delle immagini), in contrapposizione con l'utilizzo di nodi locali dotati di ridotta potenza di calcolo rispetto ai server centralizzati.

Un secondo problema è quello della coordinazione tra i dispositivi Edge e il Cloud, considerando che per ognuno di essi si hanno verosimilmente differenti capacità di calcolo e tipologie di connessione alla rete.

In ultimo, dal lato privacy, anche se le potenziali minacce sono ridotte rispetto a una soluzione unicamente basata sul Cloud Computing, la riservatezza rimane comunque un punto delicato, poiché i dati necessitano di essere condivisi tra i vari dispositivi e, pertanto, risulta necessario l'utilizzo della co-

municazione in rete, ciò unito al fatto che il dispositivo perimetrale possiede meno potenza di calcolo da dedicare a complessi algoritmi di crittazione.

Con queste premesse nasce l'idea di realizzare Edge Engine: un runtime system generico, slegato dall'hardware, in grado di interpretare codice per dispositivi multiplatforma, comprese board di sviluppo per microcontrollori. Tale sistema è in grado di elaborare i flussi di dati provenienti dai sensori ad esso collegati grazie all'utilizzo degli script: insiemi di operazioni prestabilite che possono anche essere composte al fine di eseguire calcoli complessi sui dati in ingresso. Edge Engine è configurato in modo tale da recuperare dal Cloud gli script associati al dispositivo in uso, eseguirli localmente e poi trasmettere nuovamente al Cloud i risultati ottenuti. Per il corretto funzionamento di Edge Engine è dunque necessario un server online che conservi gli script e le descrizioni dei vari dispositivi. In questo specifico caso verrà utilizzato Measurify: una piattaforma Cloud creata dall'Elios Lab dell'Università di Genova per gestire oggetti smart dell'Internet of Things (IoT).

Lo scopo del progetto in esame è la realizzazione di Edge Engine per sistemi PC Windows/ Linux/ MacOS (si veda fig. 1.3) e, consequenzialmente, l'impiego di tale motore in un contesto di realtà virtuale, al fine di illustrare e testare un'ulteriore modalità di impiego del sistema.

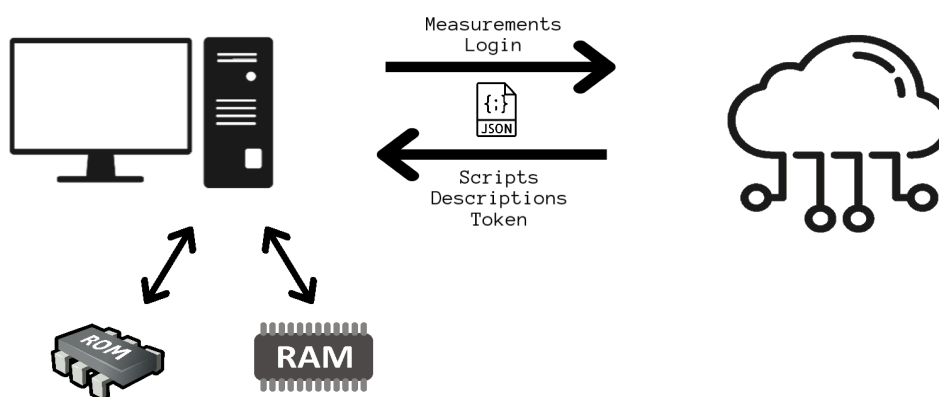


Figura 1.3: Schema funzionale di Edge Engine per PC

Il funzionamento di Edge Engine verrà inoltre analizzato nell'ambito di un progetto di tesi triennale, con l'intento di trarre da questa esperienza informazioni utili per il futuro miglioramento del sistema.

Capitolo 2

Stato dell'Arte

In un mondo in cui sempre più dispositivi necessitano di connessione alla rete e i dati personali richiedono un alto livello di protezione, oltre che di elaborazione attraverso l'impiego di algoritmi più o meno complessi, le potenzialità offerte in tale direzione dall'Edge Computing hanno attirato l'attenzione dei colossi del settore.

Google, ad esempio, ha recentemente rilasciato Edge TPU e Cloud IoT Edge [2]: il primo è un ASIC creato specificamente per eseguire l'IA a livello periferico, mentre il secondo una piattaforma per l'Edge Computing che estende le capacità di elaborazione dei dati e Machine Learning (ML) di Google Cloud ai dispositivi perimetrali. L'idea di fondo è quella di costruire i propri modelli sul Cloud, per poi utilizzarli su dispositivi Cloud IoT Edge sfruttando le potenzialità offerte dall'acceleratore hardware Edge TPU. Tale circuito è inoltre in grado di eseguire TensorFlow Lite [3]: una piattaforma che fornisce un set di strumenti i quali permettono all'utente di convertire modelli di reti neurali TensorFlow in versioni semplificate e ridotte, adatte ai dispositivi Edge [4].

Amazon, all'interno della sua offerta di servizi Cloud (AWS), mette a di-

sposizione la soluzione IoT Greengrass [5], che semplifica l'inferenza di ML in locale sui dispositivi, mediante archetipi creati, formati e ottimizzati nel Cloud. L'utente può inoltre utilizzare modelli il cui training viene fatto in prima persona. L'AWS IoT Greengrass dispone del runtime Lambda [6]: un gestore di messaggi, accesso alle risorse, ecc. I requisiti minimi a livello hardware sono 1 GHz di frequenza del processore e 128 MB di RAM.

Microsoft mette a disposizione Azure IoT Edge [7]: un servizio che permette di distribuire i carichi di lavoro del Cloud per eseguirli su dispositivi perimetrali dell'IoT. Il codice di IoT Edge supporta numerosi linguaggi, tra i quali C, C#, Java, Node.js e Python. Inoltre, la latenza è ridotta siccome i dati vengono elaborati in locale, con la possibilità di usare l'architettura hardware Microsoft, Project Brainwave [8]. I dispositivi perimetrali possono poi anche funzionare in condizioni di connessione a Internet scarsa, grazie alla gestione dei dispositivi di Azure che sincronizza in automatico lo stato più recente degli apparecchi dopo la riconnessione alla rete. Microsoft ha inoltre rilasciato EdgeML [9]: una suite di algoritmi di ML progettata per un utilizzo in situazioni di risorse ridotte. I risultati pubblicati sull'uso di EdgeML per il training su Cloud in condizioni di limitata potenza di calcolo sono indice della qualità del progetto [10]. Al momento tale libreria prevede algoritmi di tipo k-Nearest Neighbors (kNN) per classificazione, regressione e ranking noti con i nomi di Bonsai, ProtoNN e Robust PCA [11].

In ultimo, IBM ha sviluppato IBM Edge Application Manager [12]: una piattaforma intelligente, sicura e flessibile che fornisce uno strumento di gestione per l'elaborazione perimetrale. La soluzione proposta è autonoma, ossia consente ad un singolo amministratore di gestire scalabilità, variabilità e frequenza di modifica degli ambienti delle applicazioni su decine di migliaia di endpoint. Gli endpoint perimetrali si eseguono su contenitori Red Hat Open-

Shift [13]. IBM Edge Application Manager supporta inoltre tool di IA per Deep Learning e riconoscimento di voce e immagini, oltre all'analisi video e acustica.

Anche nella letteratura scientifica è possibile reperire numerosi articoli pubblicati nel periodo recente, a indicare un orientamento in direzione Edge Computing.

In particolare, riguardo la possibilità di eseguire codice multiplatforma che sfrutti le potenzialità offerte da questo approccio, si fa riferimento ad un articolo riguardante lo sviluppo di algoritmi di allocazione risorse per migliorare le prestazioni delle Vehicular Networks [14]. Per la selezione della piattaforma di esecuzione (es. Cloud Computing, Mobile Edge Computing, o Local Computing) viene utilizzato l'algoritmo kNN, mentre, per il problema di allocazione delle risorse computazionali, il Reinforcement Learning (RL). I risultati della simulazione mostrano che, rispetto all'algoritmo di base in cui tutte le attività vengono eseguite sul server di Edge Computing locale o mobile, lo schema di allocazione delle risorse consente una riduzione significativa della latenza che si attesta intorno all'80%.

Per far fronte al problema del consumo di energia dei device IoT, l'approccio Edge Computing è stato trattato all'interno di un articolo di Olli Väänänen e Timo Hämäläinen [15]. Aspetto importante di tale ricerca è la focalizzazione sul concetto di virtualizzazione. L'approccio consigliato è quello della container-based virtualization: ambienti di sviluppo virtuali emulano un determinato sistema operativo (OS), in questo modo è possibile produrre codice indipendente dalla piattaforma. I container possono essere eseguiti da dispositivi dotati di risorse limitate (es. Raspberry Pi), tuttavia richiedono una certa potenza computazionale e un generico OS.

Un ulteriore aspetto critico riguardante l'Edge Computing è il coordina-

mento tra dispositivi. Nell'ambito dei vari modelli che è possibile adottare, quelli tuple-based sono i più conosciuti e utilizzati, principalmente per la loro flessibilità [16]. Ne è un esempio il modello TuSoW (Tuple Spaces over the Web) [17]. Le tuple rappresentano i messaggi o i dati scambiati tra i componenti e l'interazione tra gli stessi è gestita definendo come e quando i vari agenti coinvolti sono in grado di inserire, leggere o elaborare i dati. Tusow permette ai vari clients di rappresentare le tuple in diversi formati: YAML, JSON, XML, FOL e plain text. Il supporto multiplatforma offerto da tale sistema mira a fornire un mezzo di interazione ad alto livello per clients eterogenei, permettendo agli sviluppatori di non curarsi delle complessità intrinseche della rete di basso livello.

La ricerca scientifica si sta orientando verso l'implementazione di algoritmi di ML applicati all'Edge Computing, come anche dimostrato dalle proposte di Google e Microsoft in tale ambito [4], [10]. È stata ad esempio testata la possibilità di eseguire algoritmi di ML su Raspberry Pi [18]. In particolare sono stati implementati tre algoritmi: Support Vector Machine (SVM), Multi-Layer Perceptron e Random Forest, raggiungendo un grado di precisione superiore all'80%, mantenendo però un basso consumo di energia.

È stato inoltre analizzato un approccio di Deep Learning in cui vengono utilizzati autoencoder a livello Edge in modo da operare una riduzione delle dimensioni dei dati [19], portando benefici in termini di tempo e spazio richiesti. Tale progetto illustra tre differenti scenari. Nel primo, dati provenienti da sensori vengono mandati a dei "nodi Edge", dove viene applicata la riduzione delle dimensioni, per poi applicare le tecniche di ML su Cloud. Nel secondo, i dati che sono stati ridotti in Edge vengono trattati sul Cloud in modo da riottenere i dati originari, per poi successivamente applicare il ML. Nel terzo ed ultimo caso invece, la tecnica utilizzata è quella del Cloud Computing: i dati

vengono mandati dai sensori direttamente al Cloud. I risultati finali mostrano come l'utilizzo di autoencoder a livello Edge riduca il numero di feature e, di conseguenza, l'ammontare di dati inviati al Cloud.

In ultimo, Respiro è un inalatore smart prodotto da Amiko [20] che contiene un processore ultra-low-power ARM Cortex-M. Tale inalatore fa affidamento sul ML per interpretare le vibrazioni provenienti dal sensore posto al suo interno. Il dispositivo è addestrato per riconoscere diverse tipologie di respiro e calcolare di conseguenza importanti parametri in ambito medico quali la capacità polmonare e la tecnica di inalazione. I dati così prodotti vengono poi processati da un'applicazione e, se la connessione lo permette, inviati al Cloud per essere conservati ed elaborati potendoli confrontare inoltre con i risultati altrui a fini statistici. Infine, attraverso un'app, viene inviato un feedback all'utente affinché possa avere sotto controllo i dati estrapolati.

Queste sono le principali soluzioni disponibili nel mercato tecnologico e nelle più recenti ricerche accademiche. Risulta evidente che gli aspetti ritenuti più importanti siano l'ottimizzazione delle risorse, il basso consumo di energia, l'indipendenza dalla piattaforma, la privacy e la bassa latenza, oltre alla facilità di utilizzo.

Edge Engine nasce con l'idea di creare un runtime system slegato dalla piattaforma che soddisfi i requisiti cardine dell'Edge Computing sopracitati. A differenza delle soluzioni appena presentate, questo progetto cerca di fornire un software completamente personalizzabile e adattabile a qualsiasi apparato IoT, come riassunto dallo schema visibile in figura 2.1.

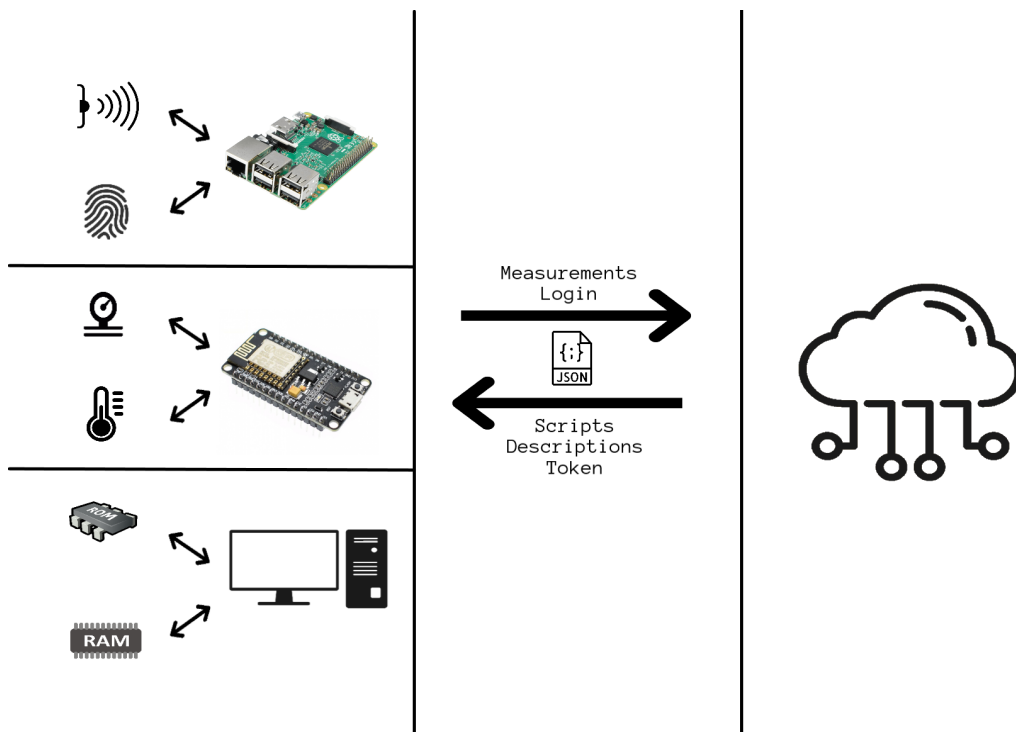


Figura 2.1: Schema funzionale di Edge Engine su piattaforme eterogenee

Capitolo 3

Contesto di sviluppo

In ambito Edge Computing, un ecosistema IoT necessita di tre componenti principali: un dispositivo che raccolga i dati dall'ambiente circostante tramite sensori, un motore di esecuzione in grado di interpretare tali informazioni e un server Cloud (si veda fig. 3.1). All'interno di quest'ultimo sono presenti le descrizioni dei device di interesse e, inoltre, vengono conservati i dati processati dall'engine.

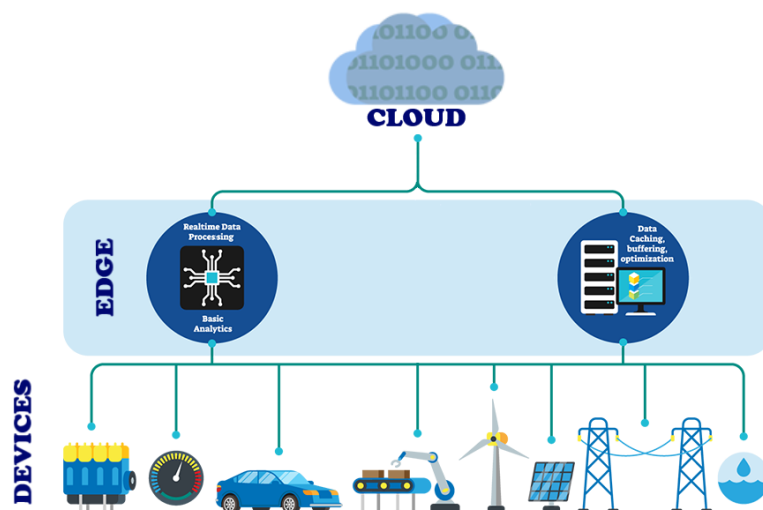


Figura 3.1: Ecosistema IoT basato sull'Edge Computing

3.1 Measurify

Measurify è il server Cloud che è stato impiegato per questo progetto: una piattaforma Cloud-based, astratta e measurement-oriented creata dall'Elios Lab dell'Università degli Studi di Genova per gestire oggetti intelligenti negli ecosistemi IoT. Measurify modella tali oggetti come risorse web esponendoli attraverso API che rispettano un'architettura REST (REpresentational State Transfer). In questo modo, l'accesso remoto a dati e risorse avviene attraverso un'interfaccia HTTP(S), indipendente dalla piattaforma, per supportare lo sviluppo di applicazioni che si servono di tali oggetti. Le risorse delle API associate al dispositivo specifico sul quale il motore è in esecuzione sono configurabili da remoto con un'applicazione client (ad esempio una web application) che le modifica per gestire l'engine.

Per poter accedere a Measurify sono necessari username ed password che permettono di ricevere un token di sicurezza. Nel caso in questione si tratta di un JSON Web Token (JWT), che andrà inserito nell'header di tutte le richieste HTTP(S) successive per garantirne l'autorizzazione.

All'interno di Measurify è presente una struttura descrittiva dei dispositivi associati al proprio username (si veda fig. 3.2). Questa è composta da diversi campi che definiscono l'oggetto in questione:

- **Thing:** è l'oggetto generico che è soggetto a misurazioni da parte dei dispositivi (ad esempio una persona, una macchina, una casa, una città, ecc.);
- **Feature:** è la grandezza fisica misurata da un dispositivo (ad esempio il battito cardiaco, la temperatura ambientale, ecc.);

- **Device:** è un'istanza del dispositivo in uso che ha una descrizione virtuale sul Cloud API ;
- **Script:** è un file che contiene informazioni su come l'Edge Engine debba manipolare, memorizzare e trasmettere al cloud gli stream di dati provenienti dai sensori o da altri dispositivi. Può trattarsi di una funzione molto complessa per gli engine più potenti oppure un semplice insieme di parametri (ad esempio la velocità di acquisizione o il numero di dati da inviare assieme) per quelli di fascia inferiore;
- **Measurement:** è il valore di una grandezza fisica misurato dal sensore di un dispositivo per uno specifico oggetto;

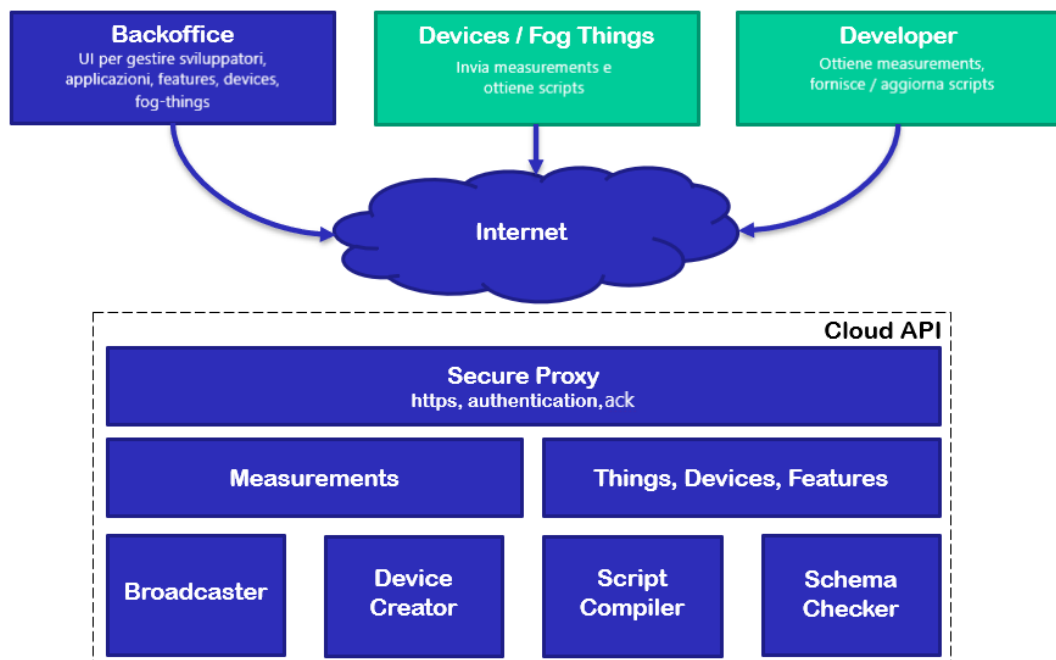


Figura 3.2: Struttura del server Cloud Measurify

3.2 Edge Engine

Edge Engine è composto da una serie di funzioni unite in un motore di esecuzione che viene eseguito su un host perimetrale. Il progetto nasce inizialmente con lo scopo di essere eseguito su piattaforme di fascia bassa. Di conseguenza, la prima versione ne prevede l'utilizzo solo su scheda di sviluppo ESP32-DevKitC V4 con modulo ESP32-WROVER-B su piattaforma Arduino, mantenendo però la possibilità di adattamento del codice a più dispositivi, come sarà mostrato nel capitolo successivo.

Il funzionamento dell'engine si può riassumere in macro-processi: accesso al Cloud, richiesta della propria descrizione virtuale, richiesta degli script da eseguire, lettura dati dai sensori, elaborazione locale dei dati, memorizzazione dei dati e invio degli stessi al Cloud.

L'engine, oltre a username e password necessari per l'accesso al Cloud, deve essere dotato di un identificativo univoco chiamato *id*, inserito all'interno del codice, che gli permetta di identificare ed accedere alla propria rappresentazione virtuale e di specificare la provenienza di tutte le informazioni che andrà a comunicare.

In seguito all'autenticazione, l'engine deve richiedere la propria rappresentazione virtuale, che ne descrive l'identità, i parametri di configurazione e le funzionalità, fornendo dunque gli identificativi degli script da eseguire. Sul Cloud, per ognuno di questi, deve essere stata in precedenza creata una risorsa di tipo *script* che potrà essere ottenuta dall'engine tramite una GET request. Gli script e la descrizione virtuale sono tutti e soli gli elementi di configurazione aggiornabili da remoto. Una volta ottenuti può iniziare la fase operativa.

L'elaborazione locale dei dati raccolti avviene quindi attraverso gli script, una composizione di un set predefinito di operazioni semplici, la cui imple-

mentazione viene precaricata nell'engine. Queste possono essere applicate agli stream di dati nell'ordine desiderato per formarne anche di complessi i quali producono in uscita nuovi stream. L'upload di questi dati sul Cloud può avvenire in due possibili modi: in modo continuo, ovvero il dato viene inviato non appena viene letto e/o processato, oppure a lotti, ovvero si può specificare il numero di misure da raggiungere perché queste vengano inviate in blocco dopo essere state memorizzate una ad una.

L'invio dei dati avviene attraverso una POST request sulla rotta delle API dedicata, fornendo nel body della chiamata, oltre al dato, anche diverse informazioni di tracciabilità, come l'identificativo del dispositivo e dello script che lo ha generato.

Durante l'esecuzione potrebbero verificarsi dei malfunzionamenti che, nel caso, devono essere comunicati dettagliatamente al Cloud, in modo da rendere noto se e quando questi si siano effettivamente verificati. Così facendo si dà anche la possibilità di prendere le dovute contromisure per correggere alcuni errori o, semplicemente, interpretare correttamente alcune situazioni inaspettate quali l'assenza prolungata di comunicazioni verso il server da parte del dispositivo, che potrebbe essere dovuta a molteplici problematiche (connessione, autenticazione fallita, server temporaneamente offline, ecc.).

In figura 3.3 è possibile vedere una struttura schematica dell'architettura del sistema Edge Engine - Measurify.

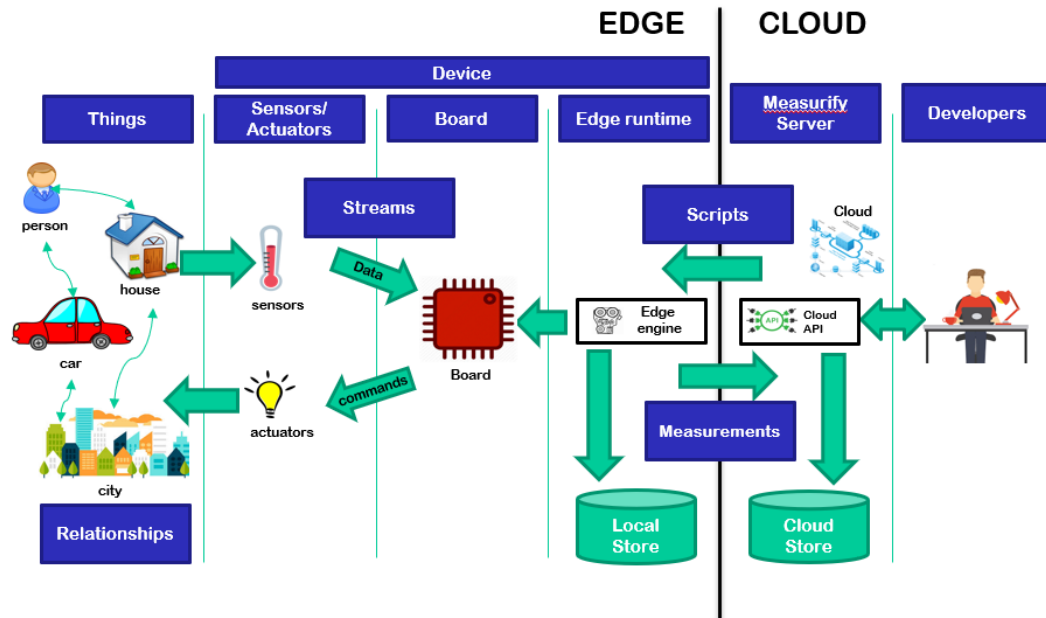


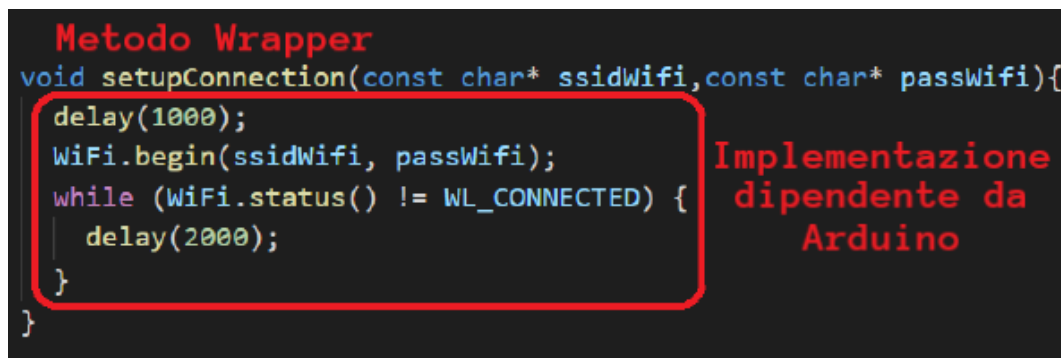
Figura 3.3: Architettura Edge Engine - Measurify

3.3 Evoluzione del progetto

A partire dal progetto Edge Engine per Arduino, l'intento è quello di realizzare un prodotto multiplatforma per dispositivi di tipo PC (Windows/ Linux/ MacOS).

La prima versione di Edge Engine nasce per essere utilizzata esclusivamente su dispositivi Arduino, pertanto, nonostante la maggior parte delle classi sia indipendente dalla piattaforma, ne restano due, **APIRest** e **Connection**, specifiche per Arduino. La prima si occupa di gestire le richieste HTTP da e verso il Cloud, mentre la seconda di gestire il modulo WiFi. Entrambe, dal momento che i task che eseguono necessitano di librerie esterne specifiche, sono giocoforza legate alla piattaforma sulla quale vengono eseguite (si veda fig. 3.4). Pertanto, il primo passo per rendere il codice multiplatforma è la

creazione di ulteriori classi wrapper `APIRest_windows` e `Connection_windows`, affinché si generalizzi sempre di più questo aspetto dell'Edge Engine.



```
Metodo Wrapper  
void setupConnection(const char* ssidWifi, const char* passWifi){  
    delay(1000);  
    WiFi.begin(ssidWifi, passWifi);  
    while (WiFi.status() != WL_CONNECTED) {  
        delay(2000);  
    }  
}
```

**Implementazione
dipendente da
Arduino**

Figura 3.4: Esempio di codice dipendente dalla piattaforma nella classe `Connection`

Inoltre, tutte le funzioni specifiche di Arduino utilizzate nel progetto devono essere sostituite da funzioni equivalenti al fine di permettere un utilizzo del sistema su dispositivi di tipo PC (ad esempio la funzione `Serial.print` è sostituita da `cout`).

Nel prossimo capitolo verrà discusso il porting di Edge Engine per dispositivi PC, mantenendo però anche la possibilità di esecuzione su piattaforme Arduino.

Capitolo 4

Implementazione

In questa sezione verrà illustrato il porting di Edge Engine da Arduino ESP32 a dispositivi di tipo PC. Al fine di mantenere il prodotto finale slegato dalla piattaforma, si è scelto di utilizzare il compilatore `g++`, disponibile su dispositivi Windows, Linux e MacOS.

In primo luogo verrà discussa l'installazione delle librerie POCO, necessarie per usufruire delle HTTP requests, per poi passare alla modifica del codice preesistente affinché possa essere eseguito sul target desiderato, mantenendo però la compatibilità con dispositivi Arduino. In ultimo, verranno mostrati un possibile esempio di utilizzo e la creazione di una libreria vera e propria, al fine di permettere una diffusione su più larga scala, unita ad una facilità d'uso maggiore.

4.1 Librerie POCO

Come anticipato in precedenza, la prima versione di Edge Engine, in quanto prevista per Arduino, è dotata di due classi wrapper, `Connection` e `APIRest`, adibite alla gestione di risorse hardware e, pertanto, legate alla piattaforma

in uso. Di conseguenza, si è resa necessaria la creazione di altre due classi che permettono di svolgere le stesse funzioni su dispositivi PC. Dal momento che le due classi necessitano di effettuare HTTP requests o di accedere alla rete Internet e l'obiettivo del progetto è quello di mantenere il più possibile il codice indipendente dalla piattaforma, si è scelto di fare affidamento sulle librerie POCO [21].

4.1.1 Installazione

Le librerie POCO sono un potente strumento C++ multipiattaforma per la creazione di applicazioni basate su rete e Internet che funzionano su desktop, server, dispositivi mobili, IoT e sistemi integrati.

L'installazione di queste librerie, tuttavia, si è rivelata più complicata del previsto, dal momento che la documentazione riguardante la compilazione delle stesse facendo uso di g++ risulta quasi del tutto assente.

VCPKG, CMake, Conan

In primo luogo, come da istruzioni del sito POCO, si è tentata l'installazione attraverso il package manager VCPKG [22]: un gestore di pacchetti open source multipiattaforma di Microsoft. Nonostante VCPKG riportasse il supporto alla compilazione di tali librerie tramite g++, il processo si interrompeva intorno al 70% a causa del seguente errore:

In file included

```
from C:/mingw/mingw64/x86_64-w64-mingw32/include/mprapi.h:16
from C:/mingw/mingw64/x86_64-w64-mingw32/include/iprtrmib.h:12,
from C:/mingw/mingw64/x86_64-w64-mingw32/include/Iphlpapi.h:15,
from C:/poco/Foundation/include/Poco/UnWindows.h:33,
```

```
from C:/poco/Foundation/include/Poco/Platform_WIN32.h:22,  
from C:/poco/Foundation/include/Poco/Foundation.h:100,  
from C:/poco/Net/include/Poco/Net/ICMPPacket.h:21,  
from C:\poco\Net\src\ICMPPacket.cpp:15:  
C:/poco/Net/include/Poco/Net/ICMPv4PacketImpl.h:72:3: error:  
expected identifier before '(' token  
    TIMESTAMP_REQUEST,  
    ~~~~~  
  
[...]  
  
mingw32-make.exe[2]: *** [Net\CMakeFiles\Net.dir\build.make:679:  
Net/CMakeFiles/Net.dir/src/ICMPPacket.cpp.obj] Error 1  
mingw32-make.exe[1]: *** [CMakeFiles\Makefile2:530:  
Net/CMakeFiles/Net.dir/all] Error 2  
mingw32-make.exe: *** [Makefile:151: all] Error 2
```

Il secondo tentativo è stato invece effettuato utilizzando CMake [23]. CMake è un tool modulare che, con poche e concise istruzioni, è in grado di generare Makefile. Esso dispone di una particolare sintassi comprensiva di moltissime macro il cui utilizzo è possibile mediante un apposito file chiamato **CMakeLists.txt**. Per la generazione del Makefile e la successiva compilazione del progetto, è necessario eseguire i seguenti comandi:

```
mkdir build  
cd build  
cmake ..  
make
```

Anche in questo caso però, la compilazione si interrompeva attorno al 70% restituendo lo stesso errore di VCPKG.

A seguito dei primi due tentativi falliti, si è scelto di provare a installare le librerie POCO tramite Conan: un package manager open source per C e C++ che consente ai team di sviluppo di gestire in modo semplice ed efficiente pacchetti e dipendenze tra piattaforme e sistemi di compilazione [24]. Il funzionamento di Conan è analogo a quello di VCPKG, ma i file necessari all'installazione delle librerie sono differenti, pertanto ci si sarebbe potuto aspettare un esito differente da quelli passati. Tuttavia, dopo aver seguito le istruzioni specificate dalla guida all'utilizzo, il risultato ottenuto è stato analogo ai precedenti.

MSYS2

In ultimo, nonostante non vi fosse alcuna documentazione a riguardo nemmeno sul sito delle librerie POCO, la mancanza di valide alternative ha portato a sperimentare una via differente, rappresentata dal tool MSYS2 [25]. MSYS2 è una raccolta di strumenti e librerie che fornisce un ambiente di facile utilizzo per la creazione, l'installazione e l'esecuzione di software nativo Windows. Essa offre build aggiornate per GCC, mingw-w64, CPython, CMake, Meson, OpenSSL, ecc. Per fornire una facile installazione dei pacchetti e un modo per mantenerli aggiornati, è dotata di un package manager chiamato Pacman [26], il quale offre molte potenti funzionalità tra cui la risoluzione delle dipendenze e semplici aggiornamenti di sistema, nonché la creazione di pacchetti. La distribuzione del software MSYS2 utilizza un porting di Pacman per creare e gestire (installare, rimuovere e aggiornare) i pacchetti binari.

Al fine di installare le librerie POCO tramite MSYS2 è necessario eseguire la seguente istruzione sul prompt dei comandi fornito in dotazione:

```
$ pacman -S mingw64/mingw-w64-x86_64-poco
```

L'installazione è finalmente riuscita, come è stato possibile verificare tramite i comandi:

```
$ pacman -Qi mingw-w64-x86_64-poco
```

```
$ pactree mingw-w64-x86_64-poco
```

Una possibile spiegazione per la quale, in questo caso, l'installazione sia andata a buon fine potrebbe essere innanzi tutto la differente origine del codice sorgente rispetto ai tool precedentemente utilizzati. Inoltre, come è stato possibile verificare direttamente sulla repository di MSYS2, il pacchetto POCO risulta aggiornato pochi giorni prima del tentativo di installazione. Ciò potrebbe indicare una possibile recente risoluzione dei problemi relativi al pacchetto che, probabilmente, non era ancora stata portata a termine nel caso degli altri package manager.

4.2 Utilizzo

Onde usufruire delle librerie POCO precedentemente installate su dispositivi di tipo PC, è necessario seguire alcuni passaggi specifici riguardo le istruzioni da dare al compilatore:

```
path\to\mingw64\bin\g++.exe
```

```
-g
```

```
path\to\EdgeEngine_library\examples\EdgineTest.cpp
```

```
path\to\EdgeEngine_library\src\connection_windows.cpp
```

```
path\to\EdgeEngine_library\src\sample.cpp
```

```
path\to\EdgeEngine_library\src\APIRest_windows.cpp
```



```
path\to\EdgeEngine_library\src\average.cpp
path\to\EdgeEngine_library\src\edgine.cpp
path\to\EdgeEngine_library\src\filter.cpp
path\to\EdgeEngine_library\src\mapVal.cpp
path\to\EdgeEngine_library\src\maxVal.cpp
path\to\EdgeEngine_library\src\median.cpp
path\to\EdgeEngine_library\src\minVal.cpp
path\to\EdgeEngine_library\src\operation.cpp
path\to\EdgeEngine_library\src\postVal.cpp
path\to\EdgeEngine_library\src\reception.cpp
path\to\EdgeEngine_library\src\script.cpp
path\to\EdgeEngine_library\src\slidingWindow.cpp
path\to\EdgeEngine_library\src\stdDeviation.cpp
path\to\EdgeEngine_library\src>window.cpp
-o
path\to\EdgeEngine_library\examples\EdgineTest.exe
-Ipath\to\msys64\mingw64\include
-Ipath\to\EdgeEngine\edge-engine\EdgeEngine_library\src
-Lpath\to\msys64\mingw64\bin
-Lpath\to\msys64\mingw64\lib
-lPocoFoundation -lPocoUtil -lPocoNet
```

Come è possibile notare, in assenza di un file `.lib`, è necessario compilare tutte le classi del progetto affinché il proprio main possa funzionare senza errori. Il file di output generato ha estensione `.exe` ed è eseguibile da linea di comando.

Le ultime cinque righe riguardano invece l'inclusione all'interno del progetto delle librerie POCO. Il comando `-I` è necessario per fornire al compilatore

la locazione degli header POCO. `-L` permette invece di inserire i percorsi aggiuntivi necessari al linker per trovare i file relativi alle librerie. In ultimo, si utilizza il comando `-l` per assegnare al compilatore i nomi delle librerie che è necessario linkare.

4.3 Classi wrapper

Come spiegato in precedenza, la prima versione di Edge Engine è dotata di due classi wrapper che svolgono compiti di connessione alla rete e invio di richieste HTTP, ossia `Connection` e `APIRest`. In questa sezione verrà discusso l'adattamento che si è reso necessario applicare affinché queste due classi potessero essere utilizzate da piattaforme di tipo PC.

L'idea iniziale era quella di modificare le classi preesistenti aggiungendo direttive `#ifdef` che permettessero di passare dalla versione Arduino a quella PC semplicemente definendo una macro. Tuttavia, dal momento che le modifiche da apportare risultavano troppo invasive, si è preferito creare due nuove classi, `Connection_windows` e `APIRest_windows`, in grado di svolgere lo stesso compito delle due originali, ma per dispositivi di tipo PC. Vedremo in seguito come sia stato reso possibile passare da una piattaforma all'altra tramite il file header `myDefines.h`

4.3.1 `Connection_windows`

La classe `Connection` si occupa unicamente di gestire la connessione WiFi attraverso alcuni metodi che permettono di collegarsi a internet sfruttando la libreria WiFi di Arduino per ESP32. L'istanza di questa classe viene creata nel setup dello sketch e gli unici parametri di cui ha bisogno per stabilire e mantenere la connessione sono le credenziali della rete. Inoltre, fornisce metodi

utili alla verifica dello stato della connessione e alla riconnessione nel caso in cui questa dovesse venire meno.

Nel caso di dispositivi PC, si ha a disposizione un'interfaccia attraverso la quale l'utente può personalmente collegarsi alla rete inserendo le credenziali richieste. Stabilire la connessione ad una rete WiFi avrebbe richiesto un'implementazione dipendente dal sistema operativo installato sulla macchina in uso, pertanto, si è presa la decisione di non implementare i metodi di connessione e riconnessione alla rete, ma soltanto di testarne lo stato. Qualora l'utente non sia connesso a nessuna rete, il programma attenderà finché questo requisito non sarà soddisfatto.

In figura 4.1 è mostrato un confronto tra l'implementazione del metodo `isConnected` per Arduino, rispetto a quello per PC. Nel caso Arduino, si può notare come il, al fine di ottenere lo stato della connessione, si faccia uso della funzione `WiFi.status`, propria della libreria WiFi di Arduino.

Nel caso invece dell'implementazione del metodo per PC, siccome utilizzando le librerie POCO non è possibile accedere direttamente allo stato della connessione senza prima effettuare una HTTP request, si è scelto di fare una GET su un sito del quale si avesse certezza di stabilità. Nel caso in cui la richiesta GET non vada a buon fine, si può assumere che il dispositivo in uso non sia connesso alla rete Internet.

ARDUINO	PC
<pre> bool connection::isConnected(){ if(!TESTING){ return (WiFi.status() == WL_CONNECTED); } else{ return connected; } } </pre>	<pre> bool connection_windows::isConnected() { try { // prepare session URI uri("https://www.google.com"); HTTPClientSession session(uri.getHost(), uri.getPort()); // prepare path string path(uri.getPathAndQuery()); if (path.empty()) path = "/"; // send request HTTPRequest req(HTTPRequest::HTTP_GET, path, HTTPMessage::HTTP_1_1); session.sendRequest(req); connected = true; cout << "You're connected" << endl; return connected; } catch (Exception &ex) { // cerr << "Error: " << ex.displayText() << endl; cout << "You're not connected" << endl; connected = false; return connected; } } </pre>

Figura 4.1: Confronto tra le due implementazioni del metodo `isConnected`

4.3.2 APIRest_windows

La classe `APIRest` è un po' più complessa: fornisce un'interfaccia semplificata per l'uso di tutte le richieste HTTP(S) (nel progetto vengono usate solo GET e POST), celando i dettagli della libreria `HTTPClient` di Arduino. La sua istanza viene utilizzata sia dalla classe principale `edgine` sia dalla classe `postVal`, ultima operazione usata in ogni script che effettua la POST dei dati quando necessario. `APIRest` espone dunque i metodi necessari alla comunicazione con il Cloud, occupandosi dei particolari implementativi della piattaforma utilizzata: preparazione dell'header e del body delle richieste, invio e gestione del responso. In seguito al fallimento di una richiesta di informazioni (GET), è l'engine a decidere come comportarsi e se comunicare o meno l'errore al Cloud; laddove invece a fallire sia la trasmissione di un dato o di un issue (POST) allora sarebbe compito di questa stessa classe memorizzare la richiesta fallita

e riprovarne l'invio fino alla corretta ricezione da parte del Cloud. Per ragioni principalmente energetiche, questi tentativi vengono fatti ogni qualvolta si renda necessario effettuare l'invio di un nuovo dato e non in maniera continuativa. In ultimo, tale classe ha il compito di calcolare e fornire la data e l'ora correnti da allegare alle misurazioni e alle comunicazioni d'errore.

Dal momento che `APIRest` fa largo uso della libreria `HTTPClient` di Arduino, si è resa necessaria una nuova implementazione della stessa per dispositivi PC, `APIRest_windows`. Questa classe ha il compito di svolgere le stesse funzioni dell'originale, ma, in questo caso, avvalendosi delle librerie `POCO`, al fine di rispettare le specifiche di compatibilità con altre piattaforme. In figura 4.2 è possibile notare le differenti implementazioni della funzione `POSTLogin`, adibita all'autenticazione sul Cloud e alla successiva ricezione del JWT.

ARDUINO	PC
<pre> string APIRest::POSTLogin (string url, string username, string password){ if(TESTING){ HTTPClient http; http.begin(uri.c_str()); //Specify the URL and certificate http.addHeader("Content-Type","application/json"); httpCode = http.POST(("{"username\":\"" + username + "\",\n" + "password\":\"" + password + "\"}").c_str()); itoa(httpCode,httpCodeTmp,10); response=string(httpCodeTmp).c_str(); if (!isHTTPCodeOk(httpCode)) { //Check for the returning code Serial.print(F("[HTTPS] POST Login... failed,")); if(httpCode<0) response=response+ " error: "+http.errorToString(httpCode).c_str(); } Serial.println(response.c_str()); http.end(); //Free the resources return response; } else // Mocking for the Unit Test { //example of token response if(username=="username" && password=="password"){ return "200[\"token\": \"JWT token\"]"; } else if(username=="username1" && password=="password1"){ return "200[\"token\": \"JWT token\"]"; } else { return "none"; } } } </pre>	<pre> string APIRest_windows::POSTLogin(string url, string username, string password) { // prepare session URI uri(url); HTTPClientSession session(uri.getHost(), uri.getPort()); // prepare path string path(uri.getPathAndQuery()); if (path.empty()) path = "/"; // send request HTTPRequest req(HTTPRequest::HTTP_POST, path); req.setContentType("application/json"); // set the body string body = ("{"username\":\"" + username + "\",\n" + "password\":\"" + password + "\"}"); req.setContentLength(body.length()); ostream fos = session.sendRequest(req); os << body; // sends the body req.write(std::cout); // print out request // get response HTTPResponse res; cout << res.getStatus() << " " << res.getReason() << endl; istream &is = session.receiveResponse(res); // StreamCopier::copyStream(is, cout); response = ""; StreamCopier::copyToString(is, response); itoa(res.getStatus(), status, 10); response = string(status) + response; cout << "RESPONSE:" << endl; cout << response << endl; // cout << isHTTPCodeOk(res.getStatus()) << endl; if (!isHTTPCodeOk(res.getStatus())) { //Check for the returning code cout << "[HTTPS] POST Login... failed, error: " << res.getStatus() << " " << res.getReason() << endl; // if(res.getStatus() < 0) // response = response + " error: " << res.getReason(); } session.abort(); return response; } </pre>

Figura 4.2: Confronto tra le due implementazioni del metodo `PostLogin`

In primo luogo, il metodo prende in ingresso tre stringhe: url del server, username e password dell'utente necessari ad ottenere i permessi per accedere alle proprie risorse. In seguito viene eseguita una POST all'interno del cui body vengono specificati username e password, scritti in formato JSON. La response contiene lo stato della richiesta HTTP, che indica se l'operazione sia andata a buon fine o meno, e il JWT necessario per gli accessi futuri alle risorse.

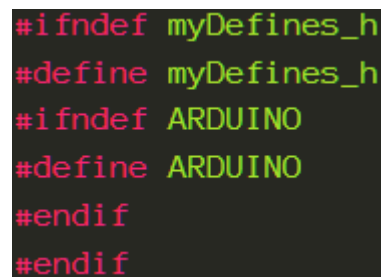
4.3.3 myDefines.h

Una volta implementate le due classi wrapper `Connection_windows` e

`APIRest_windows`, volendo mantenere la possibilità di passare da una piattaforma all'altra in modo rapido e semplice, si è scelto di creare un header file,

`myDefines.h`, attraverso il quale è possibile decidere il target di esecuzione del proprio codice senza necessità di modificare ulteriori files.

`myDefines.h` ha la struttura mostrata in figura 4.3. Essa consiste semplicemente nella definizione della macro `ARDUINO` nel caso in cui si desideri compilare per tale piattaforma. Laddove invece tale macro non sia definita, verrebbe scelta di default la compilazione per PC. `myDefines.h`, deve essere inclusa in tutti i file che utilizzano i metodi delle classi wrapper sopra descritte. All'interno di tali file, lo switch di piattaforma è gestito tramite direttive `#ifdef`, come visibile in figura 4.4.



```
#ifndef myDefines_h
#define myDefines_h
#ifdef ARDUINO
#define ARDUINO
#endif
#endif
```

Figura 4.3: `myDefines.h`

```
edgine::edgine(){  
    #ifdef ARDUINO  
    Api=APIRest::getInstance();  
    conn=connection::getInstance();  
    #else  
    Api = APIRest_windows::getInstance();  
    conn = connection_windows::getInstance();  
    #endif  
}
```

Figura 4.4: Esempio di utilizzo della direttiva `#ifdef`

In questa specifica situazione, nel caso in cui, come in figura, sia definita la macro `ARDUINO`, vengono create istanze delle classi `Connection` e `APIRest`, altrimenti delle classi `Connection_windows` e `APIRest_windows`.

4.4 Esempio di prova

In seguito all'implementazione delle classi citate in precedenza, si è deciso di sviluppare un semplice main per PC in grado di simulare un'applicazione IoT per il monitoraggio della temperatura, al fine di testare quanto realizzato.

Il device simulato possiede una feature: temperature. In prima istanza è necessario impostare tutti i parametri della descrizione virtuale e creare all'interno del Cloud la risorsa che rappresenti il dispositivo d'interesse. Dopodiché, bisogna costruire gli script che si intende eseguire sui flussi di dati e assegnarli alla risorsa precedentemente creata sulle API di tale dispositivo. È necessario inoltre aggiungere un utente (parametri username e password nel codice) sul Cloud attraverso il quale l'engine possa autenticarsi.

Il main è suddiviso in due fasi principali, associate alle funzioni `setup` e `loop` che richiamano la configurazione tipica degli sketch Arduino. Le procedure da svolgere nella funzione `setup` sono la configurazione del sensore,

l'avvio della connessione Internet e l'assegnazione dei parametri alla struttura dati `options` per inizializzare l'engine. Nella funzione `loop` invece, è disposta la creazione dei sample (connotati da feature, data e valore letto dai sensori), poi processati dall'engine tramite gli script ricevuti dal Cloud. Tali script sono eseguiti ciclicamente e il risultato delle operazioni portate a termine viene conservato sul Cloud. Il processo di interazione Cloud - Edge appena descritto è riassunto nella tabella 4.1. In questo specifico caso, siccome si tratta di un esempio di test e non di un'applicazione reale, la funzione `loop` viene eseguita una volta sola, dal momento che l'interesse non è l'invio dei dati in maniera iterativa, ma la verifica del funzionamento di un singolo script.

Già da un esempio semplice come quello appena descritto, è possibile rendersi conto di come la realizzazione di un'applicazione IoT servendosi di Edge Engine riduca in maniera drastica la quantità di codice da produrre e il tempo di sviluppo del software. Esso fornisce la possibilità non solo di eseguire in locale gli script caricati da remoto, ma anche di inviare e ricevere delle informazioni dal Cloud e gestire eventuali malfunzionamenti, comunicandone l'avvenimento al server online.

Request	Task	Descrizione
POST	Credenziali di Login	Viene ricevuto il JWT dal Cloud
GET	Descrizione del device	Gli script vengono recuperati dal Cloud
POST	Measurements	I dati vengono salvati sul Cloud

Tabella 4.1: HTTP requests generalmente eseguite da Edge Engine

4.5 Creazione della libreria

Come è stato possibile notare nella sezione 4.2, il sistema, per poter essere utilizzato, necessita della compilazione di tutte le classi del progetto, andando dunque a rendere obbligatoria anche la presenza dei file `.cpp` (oltre agli

header) all'interno del proprio ambiente di sviluppo. Questa procedura è assai utile in fase di implementazione del sistema, dal momento che in questo modo è possibile agire direttamente sulle classi in caso di errori o necessità di cambiamenti al codice. Tuttavia, non lo è altrettanto nel caso di utilizzo dell'Edge Engine da parte di sviluppatori che desiderino farne uso senza volontà di applicare modifiche al codice sorgente o qualora si desideri che la propria implementazione dei vari metodi non sia visibile a chi ne usufruisce.

Di conseguenza, si è deciso di creare un file `.lib` grazie al quale è facilitata la fruizione del sistema da parte di altri sviluppatori e, al contempo, viene nascosta la realizzazione delle classi appartenenti al progetto.

Di seguito vengono mostrati i passaggi che è necessario eseguire per ottenere il file `.lib` a partire dal codice sorgente:

```
g++ -c APIRest_windows.cpp
      -Ipath\to\EdgeEngine_library\src
      -Ipath\to\msys64\mingw64\include
g++ -c connection_windows.cpp
      -Ipath\to\EdgeEngine_library\src
      -Ipath\to\msys64\mingw64\include
```

[...]

```
g++ -c window.cpp
      -Ipath\to\EdgeEngine_library\src
      -Ipath\to\msys64\mingw64\include
```

```
ar rcs Edgine.lib APIRest_windows.o connection_windows.o [...]
      window.o
```

I primi passaggi, da ripetere per ogni classe presente all'interno del progetto, producono per ognuna di esse il relativo file in formato `.o`, necessario per la successiva creazione del `.lib`. Il comando `ar` poi, a partire da i file `.o` appena creati, genera la libreria `Edgine.lib`. Infine, per permetterne l'utilizzo all'interno del proprio progetto, sono necessarie le seguenti direttive per il compilatore:

```
-g
main.cpp
-o
main.exe
-Ipath\to\library\repository\include
-Ipath\to\msys64\mingw64\include
-Lpath\to\library\repository\lib
-Lpath\to\msys64\mingw64\lib
-lEdgine
-lPocoFoundation
-lPocoUtil
-lPocoNet
```

Capitolo 5

Esperimenti

Ultimata l'implementazione del codice, ora più facilmente fruibile grazie alla creazione del file `.lib`, il passo successivo riguarda lo sviluppo di applicazioni vere e proprie in grado di fare affidamento sulla libreria Edge Engine. In particolare, questo capitolo prende in considerazione tre esperimenti differenti: il primo tratta un esempio di utilizzo su PC Windows (più completo e significativo rispetto a quello descritto nella sezione 4.4), il secondo verte invece sulla creazione di un plugin che permetta di usufruire della libreria anche in ambiente Unity3D e, infine, il terzo descrive un vero e proprio caso applicativo portato a termine da due tesisti triennali del corso di Ingegneria Elettronica e Tecnologie dell'Informazione dell'Università degli Studi di Genova.

5.1 Applicazione Windows

L'esempio di utilizzo per Windows è stato creato al fine di mostrare e testare tutte le potenzialità offerte dall'allargamento del pool di piattaforme supportate da Edge Engine.

Come accennato brevemente nella sezione 4.4, in primo luogo è necessario riportare sul Cloud la descrizione della risorsa che si intende adottare. Più in particolare, sono da specificare i parametri mostrati nella tabella seguente:

Parametri	Nome	URL
Thing	my-pc	url/v1/things/my-pc
Feature	total-ram, total-rom, available-ram, available-rom	url/v1/features
Device	pc-probe	url/v1/devices/pc-probe
Script	total-rom-installed, total-ram-installed, ram-available, rom-available, average-hourly-available-ram, ram-available-to-mb, rom-available-to-mb, max-available-ram, max-available-rom	url/v1/scripts

url = `http://students.atmosphere.tools/`

Tabella 5.1: Parametri Measurify della risorsa Windows

Il device **pc-probe** contiene all'interno della sua descrizione le feature e gli script ad esso associati. Le prime indicano grandezze fisiche misurabili dal dispositivo, mentre i secondi rappresentano le funzioni di elaborazione che è possibile applicare ai dati ricevuti.

La tabella 5.2 mostra la struttura degli script implementati. Essi sono composti da due campi principali: `_id` e `code`. `_id` specifica il nome associato allo script stesso, mentre `code` contiene le effettive operazioni che il dispositivo fisico andrà ad effettuare.

_id	code
ram-available	<code>available-ram().send()</code>
rom-available	<code>available-rom().send()</code>
total-ram-installed	<code>total-ram().send()</code>
total-rom-installed	<code>total-rom().send()</code>
max-available-ram	<code>available-ram(10).max().send()</code>
max-available-rom	<code>available-rom(10).max().send()</code>
ram-available-to-mb	<code>available-ram().map(a*1024).send()</code>
rom-available-to-mb	<code>available-rom().map(a*1024).send()</code>
average-hourly- available-ram	<code>available-ram(6m).window(+,0,10).map(a/10).send()</code>

Tabella 5.2: Script correlati al device **pc-probe**

Prendendo ad esempio lo script **ram-available-to-mb**, il codice in oggetto permette di ricavare la quantità, espressa in gigabyte, di memoria RAM disponibile tramite l'operazione **available-ram()**, la quale viene poi concatenata a **map(a*1024)**, che converte il valore ottenuto in megabyte, moltiplicandolo per un fattore 1024. Infine, tramite la **send()**, il campione appena elaborato viene inviato a Measurify.

Lo script **average-hourly-available-ram**, invece, calcola la media oraria di RAM disponibile, ma necessita di un tempo di elaborazione più lungo di una singola esecuzione dell'engine, come specificato dal codice:

```
available-ram(6m).window(+,0,10).map(a/10).send()
```

L'operazione **available-ram(6m)** permette di campionare ogni sei minuti. Tramite la **window(+,0,10)** poi, viene presa una finestra di dieci input e calcolata la somma di tali elementi. Il calcolo della media viene poi portato a termine grazie alla **map(a/10)**, che opera una divisione sul totale per il numero di campioni presi in considerazione. Infine, il risultato viene inviato al Cloud grazie alla **send()**.

Una volta riportata su Measurify la descrizione della risorsa di cui si intende usufruire, è possibile passare allo sviluppo del codice dell'applicazione. Si

hanno nuovamente due funzioni principali, **setup** e **action**, che svolgono gli stessi task di autenticazione, elaborazione e invio dei dati descritti nella sezione 4.4. In questo caso però, **action** viene eseguita ciclicamente per un numero di volte specificato dalla variabile **loopCount**. Ciò al fine di poter disporre di un tempo di esecuzione più lungo, nel caso in cui si desideri, ad esempio, usufruire dello script **average-hourly-available-ram**, il quale richiede un'ora di tempo per la collezione dei campioni.

Inoltre, al fine di recuperare dalla macchina in uso le informazioni relative all'utilizzo delle memorie RAM e ROM, sono state implementate le funzioni **getRAMinfo** e **getROMinfo**. Esse sono esclusive per piattaforme dotate di sistema operativo Windows, in quanto fanno uso della libreria proprietaria. Qualora si riveli necessario modificare il progetto volendolo adattare ad altri OS, sarà sufficiente sostituire le suddette funzioni con altre, specifiche del target desiderato.

Un esempio di dato conservato sul Cloud, elaborato dallo script **ram-available-to-mb** del device **my-pc** è mostrato in figura 5.1.

```
{
  "visibility": "private",
  "tags": [],
  "_id": "5f0c1b8cfa2d7c7e09ae1102",
  "thing": "my-pc",
  "feature": "available-ram",
  "device": "pc-probe",
  "script": "ram-available-to-mb",
  "samples": [
    {
      "values": [
        7805.742188
      ]
    }
  ],
  "startDate": "2020-07-13T08:30:03.896Z",
  "endDate": "2020-07-13T08:30:03.896Z"
}
```

Figura 5.1: Dato relativo al valore di memoria RAM disponibile convertito in MB e conservato sul Cloud

5.2 Applicazione alla Realtà Virtuale

Edge Engine nasce dall'idea di voler offrire uno strumento modulare e slegato dall'hardware che possa facilitare e incentivare lo sviluppo di applicazioni IoT. Questo motore, oltre ad essere una valida soluzione per tale scopo, potrebbe inoltre risultare ugualmente efficace in un contesto totalmente differente, come quello dei videogiochi. Ciò è dovuto al fatto che, attualmente, non esiste uno strumento che elabori al tempo stesso dati provenienti da sorgenti così eterogenee come possono apparire un sensore fisico ed uno presente all'interno di un mondo virtuale. Si pensi per esempio ad un videogiocatore che abbia intenzione di monitorare il proprio battito cardiaco e, al contempo, processare dati provenienti dall'ambiente di gioco. Edge Engine, se adattato ad un Integrated Development Environment (IDE) dedito alla creazione di videogiochi, potrebbe essere in grado di svolgere entrambi i task richiesti, grazie all'alto livello di scalabilità di cui dispone.

Come ulteriore obiettivo, dunque, si intende trattare lo sviluppo di un'applicazione per Unity3D con lo scopo di poter così osservare il funzionamento di Edge Engine in un contesto, la realtà virtuale, diverso da quello per cui è stato inizialmente pensato, ma che può spesso rivelarsi strettamente collegato, o collegabile, all'ambiente reale.

In primo luogo verranno descritti i passi richiesti per la creazione di un plugin, necessario al fine di utilizzare Edge Engine all'interno dell'ambiente Unity3D. In seconda istanza poi, sarà discussa l'implementazione di un'applicazione esemplificativa che sfrutti il plugin in precedenza sviluppato.

5.2.1 Plugin

All'interno di Unity, normalmente si utilizzano gli script C# per la creazione di funzionalità, ma è anche possibile includere codice creato al di fuori di Unity sotto forma di plugin. I plugin sono librerie di codice nativo specifiche della piattaforma. Possono accedere a funzionalità quali chiamate del sistema operativo e librerie di codici di terze parti che altrimenti non sarebbero disponibili per Unity. Tuttavia, queste librerie non sono accessibili agli strumenti di Unity come le normali librerie utilizzabili all'interno dell'IDE.

Per la creazione del plugin di Edge Engine, che corrisponde in tutto e per tutto ad una Dynamic-Link Library (DLL), si è scelto di utilizzare Visual Studio, data l'affinità di tale IDE con Unity. Dal momento che il compilatore offerto da Visual Studio, MSVC, è differente da g++ (usato in precedenza per la creazione della libreria Edge Engine e l'installazione delle librerie POCO), è stato in primo luogo necessario ricompilare entrambi i progetti.

Per quanto riguarda le librerie POCO, in questo caso, siccome il compilatore proposto di default da VCPKG è proprio MSVC, in quanto sviluppati entrambi da Microsoft, si è deciso di utilizzare questo package manager, scegliendo la versione a 64 bit. Rispetto ai problemi riscontrati in precedenza con il compilatore g++, l'installazione delle librerie è andata a buon fine senza alcun intoppo. Per la compilazione è necessario eseguire i seguenti comandi:

```
vcpkg install poco:x64-windows
vcpkg integrate install
```

Il comando di `integrate install` non è obbligatorio, ma semplifica l'utilizzo delle librerie POCO all'interno dei progetti, evitando allo sviluppatore di dover inserire le locazioni degli header e dei file `.lib`, necessari alla compilazione.

Per quanto riguarda invece la libreria Edge Engine, è stato creato nuovamente un file `.lib`, ma questa volta, facendo affidamento a MSVC per i motivi sopracitati. Per poter poi utilizzare tale libreria all'interno del progetto di creazione della DLL, è stato necessario fornire al compilatore, tramite la schermata di impostazioni di Visual Studio, la locazione degli header e del file `Edgine.lib`.

Una volta poste le basi per la creazione della libreria dinamica che sarebbe poi stata utilizzata come plugin all'interno di Unity, è stato possibile passare alla fase di implementazione della stessa.

Lo schema della DLL che è stata prodotta prevede una singola classe, dentro la quale sono presenti le funzioni a cui si intende poi fare riferimento dal lato C# e il relativo header. All'interno di quest'ultimo, è necessario definire tali funzioni con l'attributo `__declspec(dllexport)`, il quale indica che esse possono essere chiamate dallo script che sfrutta il plugin. Inoltre, tramite la keyword `extern "C"`, si indica al compilatore che le funzioni in oggetto devono impiegare la convenzione di linking del linguaggio C. Ciò è necessario al fine di esportarle correttamente, altrimenti, a causa del mangling operato dai compilatori C++, non sarebbe poi possibile far riferimento ad esse in ambiente Unity. Il mangling è una tecnica utilizzata per distinguere funzioni che hanno lo stesso nome (overloading). Essa consiste nella manipolazione del nome in fase di compilazione affinché esso sia univoco, ma, in questo specifico caso, ciò renderebbe ignoto allo sviluppatore il nome definitivo associato e, per questo motivo, non sarebbe poi in grado di utilizzare tali funzioni all'interno del plugin. Il linguaggio C, invece, non prevede il mangling delle funzioni, dal momento che la tecnica di overload non è supportata. Pertanto, il file binario generato dalla compilazione contiene i nomi originali delle funzioni esportate, che possono essere chiamate dal lato C# senza incongruenze. Siccome la

convenzione di linking adottata è quella del linguaggio C, è necessario che, nonostante il corpo delle funzioni sia scritto in C++, i tipi di dato di input e output siano quelli propri del C.

Andando, come di consueto, a riprendere la struttura degli sketch Arduino, si è scelto di implementare due funzioni esportabili: **Setup** e **Action**. Il loro funzionamento è analogo ai casi precedentemente discussi, con un'importante differenza relativa all'inserimento delle caratteristiche descrittive, quali *device*, *thing*, *feature*, *username* e *password*. In questo specifico caso infatti, se prima tali informazioni era necessario inserirle all'interno del codice, ora si è scelto di renderle configurabili da parte dell'utente dal lato Unity. Di seguito sono mostrate le firme delle due funzioni in oggetto:

```
void Setup(char* myId, char* myPw, char* myThing, char* myDevice,  
           char* myDeviceId)  
void Action(char* sampleName, float hits)
```

La **Setup** prende in ingresso nell'ordine *username*, *password*, *thing*, *device* e *deviceId*, un identificativo univoco per ogni dispositivo.

La **Action**, invece, riceve il nome della *feature* e il relativo dato misurato per poterlo processare e inviare successivamente al Cloud.

Il progetto così compilato produce i file `NATIVECPPLIBRARY.dll`, `pcred.dll`, `PocoFoundationd.dll`, `PocoNetd.dll`, `zlibd1.dll`. Il primo rappresenta proprio la libreria dinamica contenente le funzioni appena create che vengono poi utilizzate all'interno di uno script Unity. Il secondo, invece, è una DLL di file di sistema essenziali dell'OS Windows. Infine, gli ultimi tre sono un'elaborazione, operata durante la creazione della DLL, delle librerie POCO, necessarie al funzionamento di `NATIVECPPLIBRARY.dll`. Al fine di utilizzare all'interno di Unity la DLL appena creata, è necessario copiare questi file dentro la cartella di progetto.

5.2.2 Unity

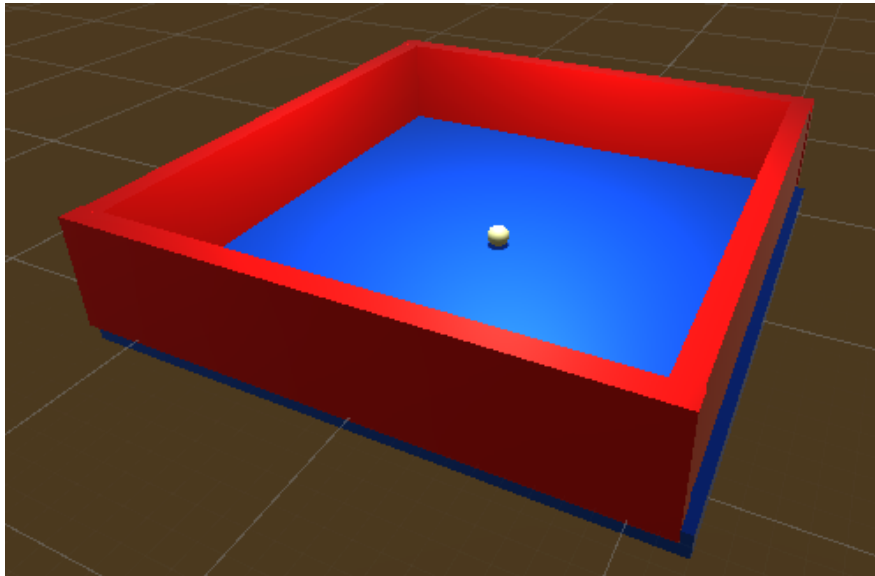


Figura 5.2: Scena di gioco Unity3D

Sul fronte Unity, avendo come obiettivo primario il test delle funzionalità di Edge Engine anche in un contesto di realtà virtuale, si è deciso di creare una semplice applicazione di esempio in grado di raccogliere dati provenienti dalla scena di gioco e, tramite il plugin descritto precedentemente, di inviarli a Measurify. In particolare, è stato creato un ring contenente al suo interno una pallina controllabile tramite i tasti freccia del PC (si veda fig. 5.2). Ogniqualvolta la pallina urta uno dei quattro muri delimitanti, la collisione viene registrata all'interno di una variabile incrementale e, in caso di raggiungimento di un numero di collisioni multiplo di cinque (lo zero è escluso per ovvi motivi), tale valore viene inviato al Cloud. Come si può facilmente dedurre, tale applicazione, diversamente da quella per Windows, rappresenta un mero esempio di utilizzo di Edge Engine all'interno dell'ambiente di gioco e non intende in

alcun modo mostrare un vero e proprio caso applicativo, dal momento che non è questo lo scopo del progetto Unity.

In primo luogo, è stato necessario inserire la descrizione della risorsa all'interno di Measurify, riassunta in tabella 5.3.

Parametri	Nome	URL
Thing	ball	url/v1/things/ball
Feature	collision	url/v1/features/collision
Device	unityDevice	url/v1/devices/unityDevice
Script	collisions-count-send	url/v1/scripts/collisions-count-send

url = `http://students.atmosphere.tools/`

Tabella 5.3: Parametri Measurify della risorsa Unity

Come spiegato in precedenza, siccome quest'applicazione è pensata per essere un semplice esempio guida, il valore associato alla sola feature `collision` viene inviato così com'è, senza alcuna elaborazione intermedia.

Veniamo dunque allo sviluppo dello script C# che permette la fruizione della libreria Edge Engine anche all'interno di Unity.

Ricordando la struttura delle funzioni esportate dal plugin mostrata nella sezione 5.2.1, le quali prevedono di ricevere in ingresso i nomi dei parametri descrittivi della risorsa, oltre alle credenziali di accesso, al fine di mantenere un elevato livello di configurabilità, si è scelto di impostare queste variabili come settabili da parte dell'utente dichiarandole `public`, come visibile in figura 5.3. Così facendo, lo svi-

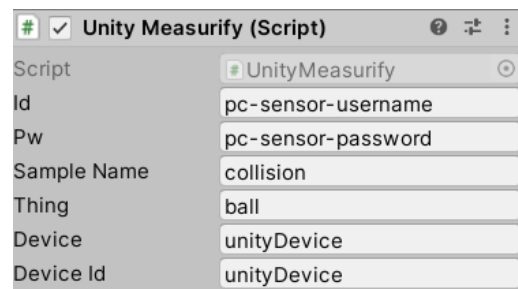


Figura 5.3: Variabili settabili dall'utente all'interno dell'inspector Unity

luppatore è in grado di personalizzare a piacere i dettagli relativi al proprio caso applicativo.

Successivamente, per poter far uso all'interno di Unity delle funzioni `Setup` e `Action`, esposte dal plugin, è necessario dichiararle con l'attributo `DLLImport`, come mostrato di seguito:

```
[DllImport("NATIVECPPLIBRARY", EntryPoint = "Setup")]
public static extern void Setup(StringBuilder myId,
StringBuilder myPw, StringBuilder myThing,
StringBuilder myDevice, StringBuilder myDeviceId);
```

```
[DllImport("NATIVECPPLIBRARY", EntryPoint = "Action")]
public static extern void Action(StringBuilder mySample,
float data);
```

- `DLLImport` indica l'importazione da una DLL;
- `EntryPoint` fornisce al compilatore il nome che la funzione ha all'interno del plugin;
- `extern` è un attributo necessario per dichiarare una funzione che è implementata esternamente.

Come è possibile notare, il passaggio dei parametri testuali avviene per lo più attraverso dati di tipo `StringBuilder`. Ciò è dovuto al fatto che l'utilizzo della memoria di `StringBuilder` e array di `char` è simile, pertanto il plugin è in grado di interpretare correttamente il contenuto del primo definito dal lato Unity.

Come di consueto, lo script Unity è organizzato nel seguente modo: un metodo `Start`, che viene eseguito una sola volta all'avvio del programma, e

uno, **Update**, che viene invece ripetuto ciclicamente. Dal momento che, anche in questo caso, lo schema di base è simile a quello degli sketch Arduino, va da sé che la funzione **Setup** verrà chiamata all'interno di **Start**, mentre la **Action** dentro **Update**, come visibile in figura 5.4.

```
// Start is called before the first frame update
void Start()
{
    idSB = new StringBuilder(id);
    pwSB = new StringBuilder(pw);
    sampleSB = new StringBuilder(sampleName);
    thingSB = new StringBuilder(thing);
    deviceSB = new StringBuilder(device);
    deviceIdSB = new StringBuilder(deviceId);

    Setup(idSB, pwSB, thingSB, deviceSB, deviceIdSB);
    threads = new Queue<Thread>();
}

// Update is called once per frame
void Update()
{
    if (MoveBall collisionCount % 5 == 0 &&
        MoveBall collisionCount != 0 &&
        oldCollisionCount != MoveBall collisionCount)
    {
        Thread t = new Thread(Play);
        threads.Enqueue(t);
        if(!threads.Peek().IsAlive)
            threads.Peek().Start();
    }
    oldCollisionCount = MoveBall collisionCount;
}
```

Figura 5.4: Metodi **Start** e **Update** dello script Unity

Start ha il compito di inizializzare tutte le variabili pubbliche di tipo **StringBuilder** coinvolte nello script. Una volta recuperati i parametri inseriti dall'utente, viene eseguita la **Setup** che, si rammenta, permette di autenticarsi su Measurify ottenendo il JWT per gli accessi successivi. In ultimo, viene inizializzato un oggetto di tipo **Queue<Thread>** che si occupa di gestire le future chiamate della funzione **Action**. La creazione di oggetti di tipo **Thread** è necessaria in quanto, altrimenti, la normale esecuzione del programma verrebbe interrotta per qualche istante ad ogni invio dei dati al Cloud, a causa del tempo necessario a compiere questo task. Grazie all'utilizzo dei thread, l'applicazione può continuare il suo normale svolgimento mentre, in parallelo, la funzione **Action** viene portata a termine. I thread vengono inseriti all'interno di una coda al fine di ottenere una più semplice gestione degli stessi, avendo la possibilità di crearli, eseguirli e distruggerli in modo ordinato facendo uso dei metodi **Enqueue**, **Peek** e **Dequeue**.

Update, innanzitutto, esegue un controllo relativo al numero di collisioni che la pallina ha effettuato. Tale valore deve essere multiplo di cinque, diverso

da zero e differente dal dato precedentemente inviato al Cloud, in modo da non avere duplicati. Se queste tre condizioni sono soddisfatte, viene creato un nuovo thread e aggiunto alla coda. A meno che il programma non ne stia già eseguendo uno precedentemente creato, il thread viene eseguito. Tutti quelli rimasti nella coda alla chiusura dell'applicazione vengono lanciati, prima dell'effettivo termine del programma, dal metodo `OnApplicationQuit`. In questo modo l'applicazione, a regime, non è sovraccarica di un eccessivo numero di thread e, al contempo, viene garantita l'esecuzione di ciascuno di essi entro la conclusione del programma, momento in cui l'utente non interagisce più con la scena di gioco.

Dall'immagine mostrata in figura 5.5 si può notare come, grazie alla creazione del plugin e all'implementazione di uno script che sfrutta tale DLL, Edge Engine sia ora in grado di comunicare correttamente con Measurify anche quando utilizzato in un contesto di realtà virtuale.

```
{
  "visibility": "private",
  "tags": [],
  "_id": "5f05e7608b005e6b0dedcdce",
  "thing": "ball",
  "feature": "collision",
  "device": "unityDevice",
  "script": "collisions-count-send",
  "samples": [
    {
      "values": [
        25
      ]
    }
  ],
  "startDate": "2020-07-08T15:33:50.492Z",
  "endDate": "2020-07-08T15:33:50.492Z"
}
```

Figura 5.5: Dato relativo al numero di collisioni effettuate dalla pallina conservato sul Cloud

5.2.3 Applicazione reale

Questa sezione, diversamente da quelle precedenti, non riguarda l'implementazione di Edge Engine su nuove piattaforme, bensì l'utilizzo di tale sistema da parte di due tesisti triennali. In particolare, il progetto in questione tratta di un rilevatore di movimento, composto da un sensore PIR HC-SR501 installato su scheda Arduino ESP8266, il quale necessita di comunicare con il Cloud Measurify per potervi inviare i dati registrati dall'ambiente circostante (si veda fig. 5.6). Le potenzialità offerte da Edge Engine risultano adatte allo sviluppo di un'applicazione del genere, siccome la gestione delle HTTP requests viene interamente gestita dall'engine, permettendo dunque agli sviluppatori di adoperarsi maggiormente sugli aspetti di collegamento sensore - board.

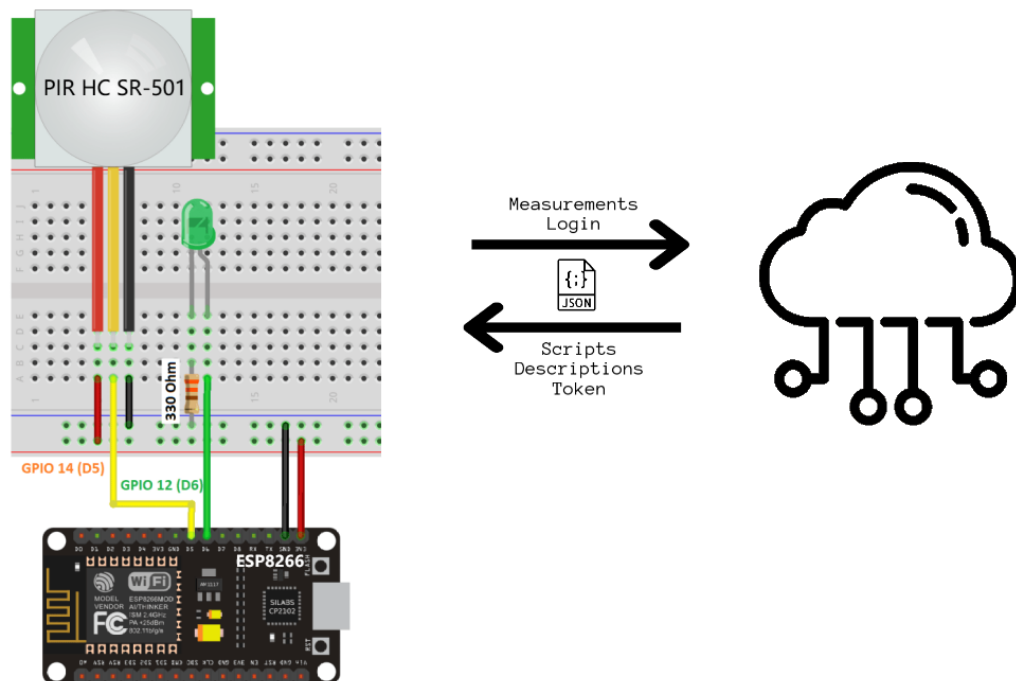


Figura 5.6: Schema funzionale di Edge Engine per Arduino

Riguardo proprio la comunicazione tra sensore e board, che avviene trami-

te Interrupt Service Routine (ISR), si è resa peraltro necessaria una piccola modifica al codice sorgente di Edge Engine. In particolare, nella funzione `getActualDate` della classe `APIRest`, chiamata proprio all'interno della ISR, veniva definita una variabile di tipo `float`. Tuttavia, l'utilizzo di tale tipologia di dato all'interno degli interrupt causa la seguente eccezione del coprocessore:

Guru Meditation Error:

Core 1 panic'ed (Coprocessor exception [...])

Ciò è dovuto all'intervento della Floating-Point Unit (FPU), non supportata all'interno della ISR. Pertanto, la soluzione adottata è stata la sostituzione del tipo `float` con `long` che, non essendo a virgola mobile, non presenta questa problematica.

Una volta risolto questo bug nel codice, è possibile comunicare correttamente con Measurify, come testimoniato dalla figura 5.7.

```
{
  "visibility": "private",
  "tags": [],
  "_id": "5eee1eb79770994272457df1",
  "thing": "my-room",
  "feature": "presence",
  "device": "presence-detector",
  "script": "simple-presence",
  "samples": [
    {
      "values": [
        1
      ]
    }
  ],
  "startDate": "2020-06-20T14:35:34.915Z",
  "endDate": "2020-06-20T14:35:34.915Z"
}
```

Figura 5.7: Dato relativo al rilevamento di presenza effettuato dal sensore PIR

Come previsto, i benefici riscontrati dall'utilizzo di Edge Engine, a detta dei diretti interessati, sono stati molteplici. In primo luogo, la facilità d'uso e installazione del sistema ha permesso di concentrarsi maggiormente sul recupero dei dati piuttosto che sugli aspetti della comunicazione Cloud, migliorando così la qualità finale del progetto, riducendo al contempo tempi e quantità di lavoro richiesti. Inoltre, gli esempi di utilizzo forniti hanno permesso di comprendere facilmente come sfruttare il motore perimetrale, fungendo da solida base per lo sviluppo del progetto in questione. Infine, la presenza di script personalizzabili con operazioni concatenabili, nonostante il numero limitato di questi, ha permesso di raggiungere gli obiettivi prefissati in modo logico e preciso.

I principali problemi riscontrati sono stati, invece, la necessità di inserire le credenziali SSID tramite codice e una certa difficoltà di comprensione dell'interazione con il Cloud. In particolare, poiché le richieste HTTP effettuate da Edge Engine sono nascoste all'interno della libreria, non vi è alcun feedback immediato per l'utente in merito alla memorizzazione dei dati su Measurify.

Da questa esperienza di progetto, nonostante vi siano problemi minori da risolvere e ampi margini di miglioramento, si evince come Edge Engine possa essere in grado di offrire una soluzione innovativa e di facile utilizzo adatta anche agli sviluppatori meno esperti che incontrano i tipici problemi relativi alla creazione di un'applicazione IoT.

Capitolo 6

Conclusioni e Lavori Futuri

L’approccio Edge Computing si sta prepotentemente facendo strada tra le altre soluzioni del genere proposte sul mercato, quali Cloud e Fog Computing. Le potenzialità offerte da questa architettura risultano al passo coi tempi in un mondo in cui sempre più dispositivi diversi tra loro richiedono la connessione alla rete con vincoli di bassa latenza e alto livello di sicurezza.

Edge Engine nasce proprio allo scopo di soddisfare tali requisiti, permettendo l’esecuzione del proprio codice su piattaforme eterogenee, lasciando inoltre allo sviluppatore piena libertà di scelta riguardo il settore d’interesse. Tale sistema può infatti essere adottato a partire da una semplice applicazione di domotica sino ad arrivare a un caso delicato come potrebbe essere il rilevamento dei parametri vitali di un paziente.

Da quanto appreso dai vari test e dall’esperienza di utilizzo dei due tesisti triennali, Edge Engine è riuscito nell’intento di semplificare lo sviluppo di nuove applicazioni IoT, offrendo un motore di esecuzione con il quale è possibile concentrarsi soltanto sulla raccolta dei dati che si intende elaborare, senza preoccuparsi della comunicazione Edge - Cloud. Inoltre, le potenzialità di cui già era dotato sono state ulteriormente aggiornate grazie al porting su

dispositivi di tipo PC e alla creazione del plugin per Unity, ampliando così il range di piattaforme impiegabili e aggiungendo, peraltro, il supporto alla realtà virtuale.

Edge Engine presenta tuttavia alcune criticità. In primo luogo, nel caso di utilizzo della versione PC, qualora non fosse presente la connessione Internet, il programma dovrebbe attendere l'intervento dell'utente, diversamente da quanto accade invece nel caso Arduino. Ciò potrebbe dar luogo a complicanze laddove non sia richiesta o possibile la presenza umana. Inoltre, il parser attuale risulta abbastanza restrittivo nella traduzione degli script in codice eseguibile. Pertanto, al fine di aumentare la semplicità di utilizzo e composizione degli stessi, sarebbe opportuno permettere all'utente una sintassi più libera, che si avvicini maggiormente al linguaggio scritto, implementando quindi un parser più sofisticato. Un ulteriore aspetto critico legato agli script riguarda l'esiguo numero delle operazioni disponibili. Attualmente, al fine di creare funzionalità avanzate, è richiesta la concatenazione delle operazioni elementari presenti. L'aggiunta di altre più complesse, come ad esempio l'elevamento a potenza o il logaritmo, potrebbe semplificare la creazione di nuovi script, andando inoltre ad aumentare le possibilità di elaborazione dei dati.

In aggiunta ai sopracitati potenziali sviluppi del codice, derivanti da criticità minori riscontrate durante l'implementazione di applicazioni basate su Edge Engine, di seguito vengono forniti ulteriori spunti che possano essere di aiuto per un'estensione delle funzionalità del sistema.

Attualmente, la funzione che viene reiterata durante lo svolgimento del programma non prevede un tempo di esecuzione pre-impostabile, ma è gestita da un counter nella versione PC e dal metodo `Update` in Unity. Esistono svariate situazioni in cui l'utente potrebbe necessitare di un tempo totale di esecuzione configurabile, in quanto non è nota a priori la durata di un singolo

ciclo. Si guardi ad esempio un'applicazione IoT di videosorveglianza, per la quale si desideri impostare uno specifico arco temporale di funzionamento di Edge Engine. In questo caso, al momento, l'utente sarebbe costretto ad avviare e interrompere manualmente il dispositivo senza poter fare altrimenti.

Un ulteriore aspetto che sarebbe importante approfondire riguarda la fusione di contesti di sviluppo differenti, al fine di ottenere applicazioni innovative sempre più complete. In questo elaborato sono stati trattati due approcci, reale e virtuale, in modo separato. Tuttavia, si pensi ad esempio al caso di un utente che desideri monitorare i propri parametri vitali (pressione, frequenza cardiaca, ecc.) e, al contempo, raccogliere ed elaborare dati provenienti dal mondo di gioco, con l'intento di correlare queste informazioni tra loro. Una funzionalità di questo genere potrebbe inoltre attirare l'attenzione degli stessi produttori di videogiochi, dal momento che permetterebbe di estrapolare dati relativi all'esperienza d'utilizzo dell'utente, utili per carpire le sensazioni che il gioco è in grado di trasmettere.

Per quanto riguarda l'idea sopracitata, ma non solo, sarebbe opportuno sviluppare algoritmi di Machine Learning in grado di essere eseguiti anche sui dispositivi perimetrali, siano essi PC o microcontrollori. Ciò potrebbe risultare d'interesse in quanto le operazioni al momento presenti non permettono una tale profondità di elaborazione dei dati.

Sono dunque numerose le possibilità di aggiornamento di Edge Engine, ciononostante il sistema oggetto della presente tesi risulta comunque completo e pronto per essere utilizzato su diverse piattaforme, imponendosi pertanto come una soluzione innovativa nell'ambito dell'Edge Computing.

Riferimenti bibliografici e sitografici

- [1] IoTedge: *L'edge computing può fare la differenza nell'analisi dei dati*,
<https://www.iotedge.it/edge-platform/ledge-computing-puo-fare-la-differenza-nellanalisi-dei-dati/>
- [2] Injong Rhee: *Bringing intelligence to the edge with Cloud IoT*,
<https://cloud.google.com/blog/products/gcp/bringing-intelligence-edge-cloud-iot>
- [3] TensorFlow Lite,
<https://www.tensorflow.org/lite>
- [4] Louis M., Azad Z., Delhadtehrani L., Gupta S.L., Warden P., Reddi V., Joshi A.: *Towards deep learning using TensorFlow Lite on RISC-V*. Workshop Comput. Archit. Res. RISC-V 2019
- [5] Amazon Web Services - AWS IoT Greengrass,
<http://www.aws.amazon.com/greengrass/ml/>
- [6] Amazon Web Services - AWS Lambda,
https://docs.aws.amazon.com/it_it/lambda/latest/dg/welcome.html
- [7] Microsoft - Azure IoT Edge,
<https://azure.microsoft.com/it-it/services/iot-edge/>

- [8] Microsoft - Project Brainwave,
<https://www.microsoft.com/en-us/research/project/project-brainwave/>
- [9] Microsoft - EdgeML, Machine learning algorithms for resource constrained devices,
<https://microsoft.github.io/EdgeML/>
- [10] Dennis D.K., Gopinath S., Gupta C., Kumar A., Kusupati A., Patil S.G., Simhadri H.V.: *EdgeML Machine Learning for Resource-Constrained Edge Devices*,
<https://github.com/Microsoft/EdgeML>
- [11] EdgeML Algorithms,
<https://github.com/Microsoft/EdgeML/wiki/Algorithms>
- [12] IBM - IBM Edge Application Manager,
<https://www.ibm.com/it-it/cloud/edge-application-manager>
- [13] Red Hat - OpenShift,
<https://www.redhat.com/it/technologies/cloud-computing/openshift>
- [14] Yaping Cui, Yingjie Liang, Ruyan Wang: *Resource Allocation Algorithm With Multi-Platform Intelligent Offloading in D2D-Enabled Vehicular Networks*, in IEEE Access (Volume: 7), pag. 21246 - 21253, 14 Febbraio 2019,
<https://ieeexplore.ieee.org/abstract/document/8642506>
- [15] Olli Väänänen, Timo Hämmäläinen: *Requirements for Energy Efficient Edge Computing: A Survey*, in Internet of Things, Smart Spaces, and Next Generation Networks and Systems, vol. 11118, 29 Settembre 2018,
https://link.springer.com/chapter/10.1007/978-3-030-01168-0_1

- [16] G. Ciatto, S. Mariani, A. Omicini, F. Zambonelli, M. Louvel: ***Twenty years of coordination technologies: State-of-the-art and perspectives***, in Coordination Models and Languages, Springer, vol. 10852, pag. 51-80, 2018
- [17] G. Ciatto, L. Rizzato, A. Omicini, S. Mariani: ***TuSoW: Tuple Spaces for Edge Computing***, 2019 28th International Conference on Computer Communication and Networks (ICCCN), Valencia, Spagna, 2019, pag. 1-6, doi: 10.1109/ICCCN.2019.8846916, <https://ieeexplore.ieee.org/abstract/document/8846916>
- [18] Yazici M.T., Basurra S., Gaber M.M.: ***Edge Machine learning: Enabling smart Internet of Things applications***. Big Data Cogn. Comput. 2018, 2, 26
- [19] Ghosh A.M., Grolinger K.: ***Deep Learning: Edge-Cloud Data Analytics for IoT. IEEE Canadian Conference of Electrical and Computer Engineering (CCECE)***, Edmonton, AB, Canada, 5-8 Maggio 2019
- [20] Brian Fuller: ***AI Technology Helping Asthma Sufferers Breathe Easier***, <http://www.hackster.io/news/ai-technology-helping-asthma-sufferers-breathe-easier-50775aa7b89f>
- [21] POCO C++ Libraries, <https://pocoproject.org/index.html>
- [22] Microsoft - vcpkg: Gestione pacchetti C++ per Windows, Linux e macOS, <https://docs.microsoft.com/it-it/cpp/build/vcpkg?view=vs-2019>
- [23] CMake, <https://cmake.org/>

- [24] Conan, the C/C++ Package Manager,
<https://conan.io/>
- [25] MSYS2 - Software Distribution and Building Platform for Windows,
<https://www.msys2.org/>
- [26] Pacman,
<https://wiki.archlinux.org/index.php/pacman>

Ringraziamenti

Ringrazio il professor Berta, per la disponibilità e la cordialità dimostrate nel corso di questi mesi.

Grazie agli amici più cari, una certezza, e ai compagni di corso, per merito dei quali questi cinque anni sono volati via leggeri.

Il ringraziamento più grande va infine alla mia famiglia, che mi ha sempre supportato e spronato a fare di più.

Luca

Ringraziamenti

Vorrei ringraziare il Professor Riccardo Berta per avermi aiutato in questo
lavoro di tesi.

Ringrazio la mia famiglia e i miei amici che mi sono stati accanto durante
tutto il percorso universitario.

Infine un ringraziamento speciale va a Giulia, che è sempre pronta a
incoraggiarmi e a sostenermi.

Andrea