



UNIVERSITÀ POLITECNICA DELLE MARCHE
FACOLTÀ DI INGEGNERIA

Corso di Laurea in Ingegneria Elettronica

**Studio e Sviluppo di soluzioni di interoperabilità
basate su Cloud AWS**

**Study and Deployment of interoperability
solutions based on Cloud AWS**

Relatore:

Prof.ssa PAOLA PIERLEONI

Tesi di Laurea di:

LUCA FORMICA

A.A. 2020/2021

Introduzione

Al giorno d'oggi Internet ha assunto sempre più importanza nelle nostre vite, e questo ha portato ad estendere il suo concetto ad una molteplicità di contesti, spingendo l'interesse e le necessità tecnologiche ad un collegamento più fitto tra i vari sistemi.

L'accesso ad Internet si sta espandendo sempre più agli “oggetti” dando vita al cosiddetto “*Internet of Things*” (IoT). Col termine “*Things*” (cose, oggetti), si vuole far capire appunto non solo l'interesse dell'Internet ai soli dispositivi come computer o smartphone, ma anche e soprattutto a oggetti come elettrodomestici, sensori e altri device di uso comune e quotidiano. L'Internet delle cose, quindi, rappresenta un concetto generale che racchiude un ecosistema in cui una grande varietà di dispositivi prendono dati dall'esterno e li scambiano e processano per raggiungere un determinato obiettivo. Lo sviluppo dei dispositivi IoT hanno permesso la creazione e la propagazione di sistemi per la domotica, industria, medicina ecc.. e facilitando l'interazione tra uomo e contesto di utilizzo.

Uno strumento fondamentale per questa interazione è il controllo vocale che offre un accesso comodo e intuitivo all'utente facilitando la comunicazione con la macchina. Il riconoscimento vocale nei vari sistemi IoT migliora quindi l'usabilità fornendo nuovi modi d'utilizzo di oggetti.

Lo scopo di questa tesi è la realizzazione di **SmartLight**, un sistema di controllo vocale di un dispositivo IoT in grado di cambiare lo stato di un LED grazie all'utilizzo di un assistente vocale.

L'interazione vocale con l'Alexa Service permette di inviare comandi per cambiare lo status dello smart object associato al proprio Cloud. Un aspetto importante presente in questo contesto rappresenta la vastità di ambiti di utilizzo estesa ad una grande varietà di dispositivi. In alcuni di questi ambiti si dà importanza alla questione velocità di trasferimento delle informazioni, e a tal scopo SmartLight è stato progettato per garantire un certo tipo di servizio.

Il sistema è stato sviluppato tramite l'utilizzo del Cloud di Amazon Web Service, Alexa Voice Service e la scheda ESP32.

Il seguente documento è organizzato come descritto. La prima parte introduce l'IoT descrivendone gli ambiti di utilizzo e le tecnologie a suo servizio. Nella seconda parte è rappresentata nel dettaglio la progettazione e l'implementazione. Nella terza e ultima parte vengono analizzati i test e i risultati relativi sia al funzionamento del sistema, sia ai tempi di viaggio dell'informazione.

Indice

STATO DELL'ARTE.....	1
INTERNET OF THINGS.....	3
1.1 <i>Definizione</i>	3
1.2 <i>Applicazioni dell'IoT</i>	5
1.2.2 Smart Metering	6
1.2.3 Smart Car	6
1.2.4 Smart City	7
1.2.5 Industrial IoT.....	7
1.2.6 Smart Agricolture	8
CLOUD COMPUTING.....	9
2.1 <i>Amazon Web Services</i>	10
2.1.1 IoT Core	10
2.1.2 Lambda	12
2.1.3 Identity and Access Management.....	12
ASSISTENTE VOCALE	14
3.1 <i>Tecnologia</i>	14
3.1.1 Intelligenza artificiale.....	15
3.1.2 Riconoscimento vocale.....	15
3.1.2 Apprendimento automatico	15
3.2 <i>Assistenti vocali popolari</i>	16
3.2.1 Alexa	17
3.2.2 Siri	20
3.2.3 Google Assistant.....	20
3.2.4 Cortana	20
3.2.5 Bixby	21
PROGETTAZIONE E IMPLEMENTAZIONE.....	23
PROGETTO	25
4.1 <i>Requisiti e obiettivi</i>	25
4.2 <i>Architettura</i>	26
4.2.1 Endpoint Alexa (App o Amazon Echo).....	27
4.2.2 Alexa Skill	27
4.2.3 Lambda Function	28
4.2.4 AWS IoT Core.....	29
4.2.5 MQTT Topic	29
4.2.6 Arduino Code	30
4.2.7 ESP32	31
IMPLEMENTAZIONE	34
5.1 <i>Flusso di interazione</i>	34
5.2 <i>Registrazione del dispositivo nel Cloud</i>	35
5.3 <i>Arduino sketch</i>	36
5.4 <i>Alexa custom Skill</i>	37
5.5 <i>Funzione Lambda</i>	39
5.5.1 Lambda Handler.....	39
5.5.2 Intent Handler.....	40
5.5.3 Response Builder	41
RISULTATI.....	44
RISULTATI	46
6.1 <i>Ambito di funzionamento</i>	46
6.2 <i>Configurazione</i>	47
6.3 <i>ASR Delay</i>	48

6.4 <i>Trip Delay</i>	48
6.5 <i>Risultati</i>	51
CONCLUSIONI.....	59
APPENDICE A.....	61
APPENDICE B.....	67
APPENDICE C.....	70
BIBLIOGRAFIA.....	72

Indice delle figure

Figura 1 – Connessioni AWS IoT	11
Figura 2 – Schema funzionamento AWS IAM	13
Figura 3 – Interazione vocale con Amazon Echo.....	17
Figura 4 – Architettura dell’interazione vocale con l’Alexa Skill.....	19
Figura 5 – Architettura Sistema IoT progettato.....	26
Figura 6 – Invocazione Skill e interazione con i comandi	28
Figura 7 – ESP32.....	31
Figura 8 – Caratteristiche ESP32.....	32
Figura 9 – Flusso di interazione tra Publisher, Broker e Subscriber	49
Figura 10 -.- Verifica connessione da monitor seriale.....	51
Figura 11 – Feedback dell’avvenuta accensione della luce tramite pulsante.....	52
Figura 12 – Pubblicazione avviso di azionamento dell’interruttore.....	52
Figura 13 – Spegnimento luce lato Cloud	52
Figura 14 – Feedback dell’azione dal monitor seriale.....	53
Figura 15 – Feedback dell’azione dal tester MQTT	53
Figura 16 – Interazione vocale con Alexa tramite il tester dell’Alexa Developer Console	53
Figura 17 – Input e Output in formato JSON generate da Alexa	54
Figura 18 - Ritardi di comunicazione	56
Figura 19 - Round Trip Delay con scarto quadratico medio	57

Parte I

Stato dell'arte

Capitolo 1

Internet of Things

Nel mondo ormai il numero di dispositivi connessi ad Internet supera di gran lunga la totalità degli abitanti della terra e si stima che, entro il 2025, si avranno più di 30.9 miliardi di connessioni [1]. In una situazione del genere, l'Internet of Things sta acquistando molto velocemente importanza ma soprattutto utilità nella vita quotidiana. L'idea di base dell'IoT è quella di avere oggetti in grado di interagire e di cooperare tra loro al fine di eseguire determinate funzioni.

1.1 Definizione

Con Internet of Things nel 1999 Kevin Ashton si riferiva all'uso dei tag RFID (Radio Frequency Identification) [2] per tracciare dei prodotti presenti in un inventario. Successivamente nel 2005 questo termine viene ufficializzato dalla International Telecommunication Union, specificando che le cose ("Things") non sono solo i tag RFID ma rappresentano diversi oggetti in grado di interagire con il mondo esterno tramite la connessione ad Internet. Questo è stato un punto di partenza per la IEEE che ha dato la definizione esatta dell'IoT [3], cioè una rete di oggetti - ognuno dotato di sensori - collegati ad Internet. Gli oggetti ("Things") sono visti come partecipanti

attivi nei processi informativi grazie alla capacità di interagire e comunicare tra loro e con l'ambiente esterno, scambiando dati ed informazioni raccolti dall'ambiente circostante. Inoltre, sono in grado di reagire agli eventi del mondo reale e di influenzarlo eseguendo azioni o creando servizi, senza necessariamente richiedere l'intervento dell'uomo.

Ci si chiede però che cosa rende un oggetto "smart" [4] e per questo si ha bisogno di due proprietà fondamentali:

- Identificazione: l'oggetto deve essere dotato di un identificativo univoco nel mondo digitale (una sorta di indirizzo IP, esattamente come una pagina web nell'Internet tradizionale che tutti conosciamo).
- Connessione.

Ci sono ovviamente altre proprietà di base, che dipendono però dal contesto di utilizzo: stato di funzionamento, localizzazione, tracciabilità, sensing e metering.

In definitiva si può dire che un sistema IoT deve avere i seguenti requisiti [5]:

- Eterogeneità: l'IoT è caratterizzata da una forte eterogeneità in termini di varietà di device, tecnologie, servizi e campi applicativi.
- Scalabilità: il crescente numero di dispositivi collegati ad un'infrastruttura globale porta il problema della scalabilità su diversi livelli, cioè: livello architettura, livello di identificazione/indirizzamento degli oggetti, livello di comunicazione e livello di mapping oggetti/servizi.

- **Autonomia:** data la complessità e la varietà degli scenari possibili nell'IoT, gli oggetti devono possedere capacità di adattamento, di auto-configurazione, elaborazione indipendente dei dati e reazione autonoma in base agli stimoli, al fine di minimizzare l'intervento umano.
- **Minimizzazione dei costi:** ottimizzare i costi del mantenimento del sistema IoT con una particolare attenzione al consumo energetico.
- **Qualità del servizio:** la garanzia di un'appropriata qualità del servizio è necessaria per servizi e applicazioni caratterizzati da un traffico dati real-time.
- **Sicurezza:** l'IoT deve garantire un ambiente sicuro in termini di sicurezza delle comunicazioni, autenticazione, integrità dei dati e privacy degli utenti.

1.2 Applicazioni dell'IoT

L'Internet of Things è uno strumento che potenzialmente ha infinite applicazioni possibili [2]: dall'autovettura che dialoga con l'infrastruttura stradale per prevenire incidenti, agli elettrodomestici di casa che si coordinano per ottimizzare l'impiego di potenza; dagli impianti di produzione che scambiano dati con i manufatti per la gestione del loro ciclo di vita; dai dispositivi medicali che si localizzano nel presidio di un pronto soccorso, agli sci che inviano informazioni sullo stato della neve, o sulla severità di una caduta. Il processo di scambio di informazioni tra gli oggetti smart, verso l'esterno e verso di loro non avviene in tutti gli ambiti con la stessa velocità: questo dipende dall'esistenza di soluzioni tecnologiche consolidate, dagli equilibri competitivi in un determinato mercato e, in definitiva,

dal bilancio tra il valore dell'informazione e il costo di creazione della rete di oggetti intelligenti.

1.2.1 Smart Home

Lo Smart Home è sicuramente il campo di applicazione più conosciuto dell'Internet of Things, in cui molteplici oggetti di uso comune, tra cui elettrodomestici, lampadine o TV possono collegarsi ad Internet per fornire servizi ed essere controllati in maniera comoda e veloce: ogni dispositivo può gestirsi in maniera autonoma senza l'intervento umano.

1.2.2 Smart Metering

Una rete di sensori posta in determinati luoghi può fornire una serie di dati utili per il monitoraggio delle attività naturali (ad esempio sismiche o idrogeologiche) per la prevenzione di disastri ambientali: l'innalzamento eccessivo del livello dell'acqua di un corso d'acqua o il rilevamento di gas possono allertare in tempi brevi chi di dovere per una gestione ottimale delle eventuali emergenze, o azionare in automatico sistemi di sicurezza. Inoltre, l'impiego di questo sistema può essere utile per la gestione dei consumi di gas, acqua o energia elettrica tramite i contatori connessi.

1.2.3 Smart Car

Negli ultimi anni sono state sviluppate diverse soluzioni per la guida intelligente tramite l'utilizzo di Smart Car, dotate di sensori in grado di fornire in tempo reale informazioni riguardo l'ambiente circostante e permettendo così un'esperienza di guida più sicura tramite l'attivazione del "collision

avoidance” che interviene automaticamente sulla frenata o sulla correzione della traiettoria, evitando di causare incidenti. Altre soluzioni sono fornite dai sistemi GPS per la rilevazione del traffico volta a pianificare in maniera ottimale il viaggio o sensori di riconoscimento facciale per rilevare lo stato di stanchezza del conducente.

1.2.4 Smart City

Il monitoraggio e la gestione degli elementi di una città (ad esempio mezzi per il trasporto pubblico, illuminazione pubblica e parcheggi) e dell’ambiente circostante possono migliorarne vivibilità, sostenibilità e competitività. L’osservatorio dell’IoT ha rilevato che in Italia il 42% dei comuni con popolazione superiore ai 15.000 abitanti ha avviato almeno un progetto Smart City negli ultimi tre anni.

1.2.5 Industrial IoT

L’applicazione dell’IoT in questo ambito aiuta a gestire la catena di distribuzione delle varie attività logistiche delle aziende con l’obiettivo di controllarne le prestazioni e migliorarne l’efficienza. Questo è sinonimo di innovazione dei sistemi di stoccaggio, movimentazione e trasporto: gli “scaffali intelligenti” e prodotti muniti di tag RFID permettono di ridurre lo spreco di risorse analizzando i dati forniti dai dispositivi.

1.2.6 Smart Agricolture

Il monitoraggio dei parametri microclimatici a supporto dell'agricoltura aiutano a migliorare la qualità dei prodotti, ridurre le risorse utilizzate e l'impatto ambientale.

Capitolo 2

Cloud Computing

Dal sito del NIST [6], il Cloud Computing è un modello per consentire l'accesso on-demand a un pool condiviso di risorse informatiche configurabili (ad es. reti, server, storage, applicazioni e servizi) di cui è possibile eseguire rapidamente il provisioning e il rilascio con il minimo sforzo di gestione o interazione del fornitore di servizi. Il termine “Cloud” deriva dal fatto che Internet è inteso come una nuvola in cui risiedono i nostri dati. Gli oggetti sono largamente distribuiti e spesso hanno limiti che riguardano la disponibilità di memoria per lo storage, capacità di calcolo, performance, sicurezza e privacy [7]. Questi limiti sono superati grazie al Cloud Computing che offre una capacità di memoria virtuale illimitata garantendo performance migliori grazie alla condivisione delle risorse, le quali possono essere velocemente invocate, gestite ed essere eventualmente pagate secondo il loro utilizzo effettivo. Inoltre, il Cloud Computing è indipendente e permette l'accesso ai suoi servizi ovunque e con qualsiasi dispositivo che abbia accesso ad Internet.

2.1 Amazon Web Services

Amazon Web Services (nota con la sigla AWS) è un'azienda statunitense di proprietà del gruppo Amazon [7] che fornisce servizi di cloud computing su un'omonima piattaforma on demand.

Questi servizi sono operativi in 25 regioni geografiche in cui Amazon stessa ha suddiviso il globo. AWS offre oltre 150 prodotti, tra i quali AWS IoT Core e Lambda. A partire dai dati pubblicati nel quarto trimestre del 2018, AWS rappresenta per Amazon il 58% dei guadagni totali, rendendola la sua più grande fonte di incassi.

Amazon Web Services (AWS) è la piattaforma cloud più completa e utilizzata del mondo [8]. Milioni di clienti, incluse le start-up in più rapida crescita, le più grandi aziende e le agenzie governative leader di settore, utilizzano AWS per diminuire i costi, diventare più agili e innovarsi in modo più rapido. Tra i vari servizi offerti dalla piattaforma se ne vedranno solo alcuni fondamentali per la creazione di un sistema IoT basilare:

- IoT Core
- Lambda
- Identify and Access Management

2.1.1 IoT Core

AWS IoT Core è un servizio che consente di collegare miliardi di dispositivi IoT e instradare migliaia di miliardi di messaggi ai servizi AWS senza gestire l'infrastruttura [9]. In altre parole, permette di integrare diversi dispo-

sitivi IoT in soluzioni basate sul servizio stesso: lo Smart Object viene collegato a IoT Core per comunicare con i vari servizi disponibili (Lambda, S3, CloudWatch ecc..).

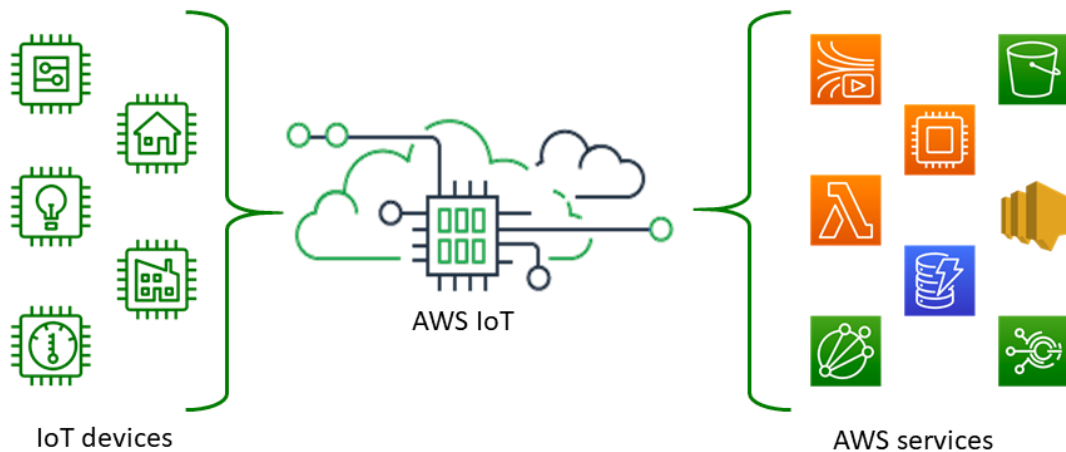


Figura 1 – Connessioni AWS IoT

AWS IoT Core supporta i seguenti protocolli:

- MQTT
- MQTT su WSS
- HTTPS
- LoRaWAN

Il broker di messaggi AWS IoT Core supporta dispositivi e client che utilizzano MQTT e MQTT su protocolli WSS per pubblicare e effettuare sottoscrizioni ai messaggi. Supporta inoltre dispositivi e client che utilizzano il protocollo HTTPS per pubblicare messaggi.

2.1.2 Lambda

AWS Lambda è un servizio di calcolo che consente di eseguire il codice senza provisioning o gestione di server [10]. Lambda esegue il codice su un'infrastruttura di calcolo ad alta disponibilità e gestisce tutta l'amministrazione delle risorse di elaborazione, compresa la manutenzione del server e del sistema operativo, il provisioning della capacità e la scalabilità automatica, il monitoraggio e la registrazione del codice. Con Lambda, è possibile eseguire il codice praticamente per qualsiasi tipo di applicazione o servizio back-end. Può essere inteso come il ponte tra un dispositivo IoT e un servizio che gestisce le richieste in input. Il codice è organizzato in funzioni e può essere basato principalmente su Python, NodeJS, Java e C#. Lambda esegue la funzione solo quando è necessario e si dimensiona automaticamente, da poche richieste al giorno a migliaia al secondo. È possibile richiamare le funzioni Lambda utilizzando l'API Lambda, oppure Lambda può eseguire le funzioni in risposta agli eventi da altri servizi AWS. Ad esempio, si può utilizzare Lambda per costruire trigger di elaborazione dati per servizi AWS.

2.1.3 Identity and Access Management

AWS Identity and Access Management (IAM) fornisce un controllo granulare degli accessi in tutto AWS [11]. Questo permette di specificare chi può effettuare l'accesso a servizi e risorse e secondo quali condizioni.

Per capire meglio il concetto di IAM si assume il seguente scenario: un CEO di un'azienda ha accesso ad AWS tramite il suo indirizzo e-mail e la sua password, e deve permettere ai suoi dipendenti di utilizzare lo stesso

servizio. Data l'impossibilità da parte del CEO di condividere le sue credenziali con gli altri (per ovvi motivi di protezione dei dati), esso crea degli utenti IAM ai quali assegna dei diritti e privilegi (Roles).

Nella figura seguente è rappresentato in maniera schematica il funzionamento di IAM: il root può decidere chi può accedere a cosa, specificando autorizzazioni granulari. Successivamente IAM applica queste autorizzazioni per ogni richiesta. L'accesso viene negato di default e viene consentito solo quando le autorizzazioni specificano un "Consenti."

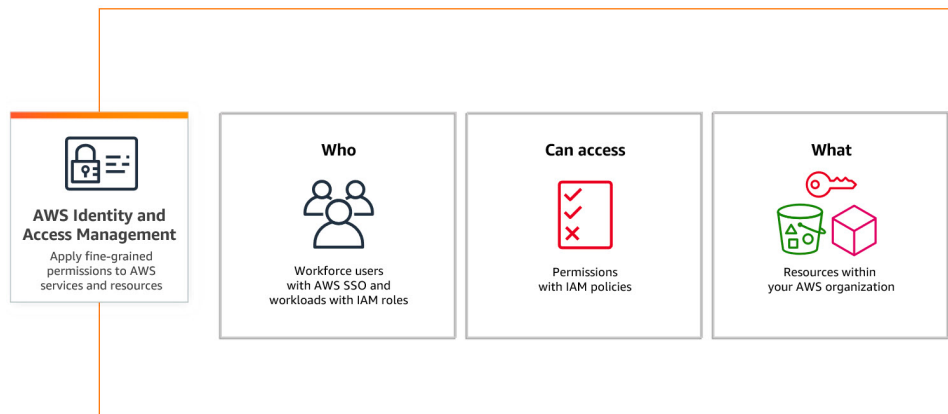


Figura 2 – Schema funzionamento AWS IAM

Capitolo 3

Assistente vocale

Un assistente vocale è un software che interpreta il linguaggio umano e, se opportunamente addestrato, può interagire con degli utenti allo scopo di fornire informazioni o compiere determinate operazioni tramite appunto dei comandi vocali [12]. La maggior parte degli assistenti vocali sono in grado di rispondere attraverso voci sintetizzate a richieste specifiche, come ad esempio la consultazione dell'orario, della temperatura esterna o l'avvenuta operazione specifica richiesta portata a buon fine. Quindi gli utenti possono porre domande ai loro assistenti, controllare degli smart objects, riprodurre un contenuto multimediale tramite la voce, gestire attività quotidiane come l'e-mail, le liste e i calendari con comandi vocali.

3.1 Tecnologia

L'assistente vocale per rispondere interagire in modo accurato alle richieste date in input da un utente, utilizza principalmente tre strumenti:

- Intelligenza artificiale
- Riconoscimento vocale
- Apprendimento automatico

3.1.1 Intelligenza artificiale

L'intelligenza artificiale utilizza algoritmi di simulazione per replicare l'intelligenza umana.

Nel 1950, Alan Turing pubblicò il suo articolo "Computing Machinery and Intelligence" [13] dove per primo poneva la domanda: "le macchine possono pensare?".

L'intelligenza artificiale moderna è generalmente vista come un sistema informatico progettato per svolgere attività che in genere richiedono l'interazione umana. Questi sistemi possono migliorarsi utilizzando un processo noto come apprendimento automatico (Machine Learning).

3.1.2 Riconoscimento vocale

Il riconoscimento vocale si basa sul segnale analogico (voce) in input per trasformarla in un segnale digitale. Dopodiché il computer lo elabora abbinandolo a parole e frasi per riconoscere il contenuto di ciò che è stato appreso. Per rendere possibile tutto questo, il computer ha a disposizione un database di parole in diverse lingue, in modo che la corrispondenza tra ciò che è stato dato in input e ciò che deve aver capito sia più accurata possibile.

3.1.2 Apprendimento automatico

L'apprendimento automatico è un sottoinsieme dell'intelligenza artificiale in cui i programmi vengono creati senza l'intervento umano: invece di scrivere il programma completo da soli, i programmatori danno degli "schemi" di intelligenza artificiale da riconoscere e da cui imparare e quindi danno

all'intelligenza artificiale grandi quantità di dati da esaminare e studiare. Insomma, invece di avere regole specifiche da rispettare, l'IA cerca modelli all'interno di questi dati e li usa per migliorare le sue funzioni già esistenti. È un processo di training, cioè di allenamento costante delle sue capacità.

3.2 Assistenti vocali popolari

Ad oggi sono diversi gli assistenti vocali sviluppati al meglio per soddisfare le varie esigenze degli utenti. Tra i più conosciuti [14] ci sono:

- Alexa
- Siri
- Google Assistant
- Cortana
- Bixby

Assistenti virtuali come questi possono fare qualsiasi cosa, dal rispondere alle domande, lanciare una moneta, riprodurre musica e controllare i dispositivi della casa come luci, termostati e serrature. Possono rispondere a molti comandi vocali, inviare messaggi di testo, effettuare telefonate e impostare promemoria, sveglie e timer.

Gli assistenti virtuali hanno la grande capacità di imparare nel tempo, conoscendo le abitudini e preferenze dell'utente che ne fanno uso. Grazie all'intelligenza artificiale, questi assistenti possono comprendere il linguaggio naturale, riconoscere i volti, identificare gli oggetti e comunicare con altri dispositivi e software intelligenti.

3.2.1 Alexa

Alexa è un'intelligenza artificiale basata su cloud di Amazon che si presenta sotto forma di assistente vocale, un software che può essere integrato in speaker e altri dispositivi smart. Questa, interpretando il linguaggio naturale, riesce ad interagire con l'uomo.

Creata da Amazon nel 2014, Alexa è stata rilasciata insieme alla linea di altoparlanti intelligenti Amazon Echo, che ruotano sulla piattaforma Alexa per rendere possibile l'interazione con l'ecosistema Amazon e consentire la connessione con una molteplicità di dispositivi intelligenti.

Alexa basa il suo funzionamento sulle skill, ovvero programmi che vengono richiamati per effettuare delle funzioni ben precise. Molte di esse sono fornite da Amazon e sono le più basilari, volte alla riproduzione di un brano richiesto, allo shopping online, alla consultazione delle condizioni climatiche ecc...

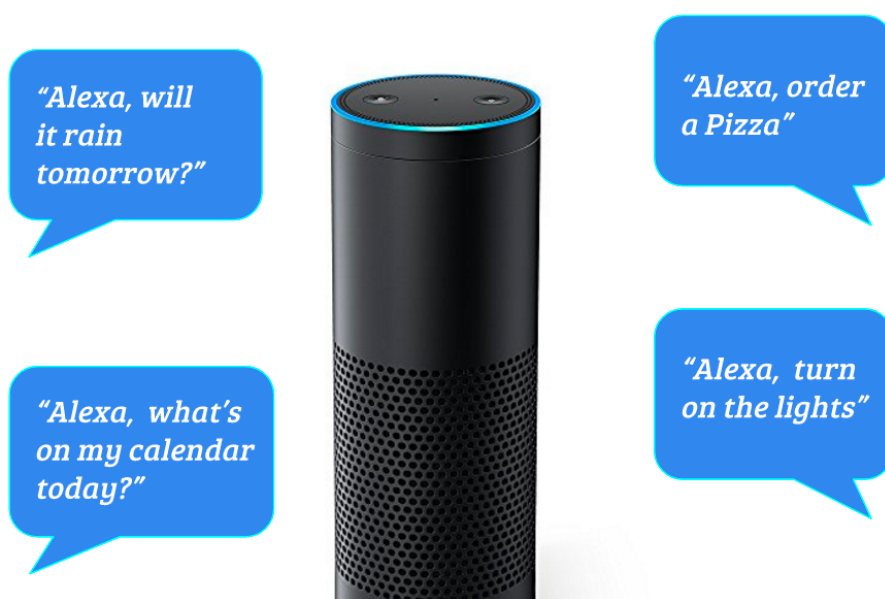


Figura 3 – Interazione vocale con Amazon Echo

Visto che si tratta di un servizio vocale implementato nel cloud, Alexa diventa sempre più intelligente per favorire una migliore esperienza dell'utente che la utilizza.

Amazon mette a disposizione uno strumento di sviluppo che è l'*Alexa Skill Kit*, tramite il quale è possibile creare nuove skill per ogni esigenza [15]. Queste skill possono essere create tramite L'*Alexa Developer Console*, nella quale è possibile scrivere un codice di tipo "Alexa-Hosted" cioè ospitato direttamente sulla piattaforma. Quindi tutto ciò che riguarda l'invocazione della skill e la sua esecuzione avviene nel Cloud di Amazon, e questo significa che nessun software deve essere installato nel dispositivo, ma basta una semplice connessione al servizio presente sul Cloud.

Per invocare una Alexa skill non serve per forza un dispositivo tipo Echo: tramite l'Alexa Voice Service (AVS) è possibile trasformare qualsiasi device provvisto di microfono, speaker e connessione ad Internet, in un prodotto in grado di utilizzare a pieno le potenzialità di Alexa.

Custom Skills

Come già accennato è possibile creare nuove skill dalla developer console di Alexa. Quando si crea una nuova skill si creano:

- Una serie di *intents* che rappresentano le azioni che la skill può effettuare. Sono il cuore della skill.
- Le *sample utterances* che specificano le frasi e le parole che l'utente deve pronunciare per invocare gli *intents*.
- Un *invocation name* che identifica la skill. L'utente farà uso di questo nome per iniziare la conversazione con la skill.

- Una configurazione che mette insieme tutte le cose appena elencate per permettere ad Alexa di instradare le richieste al servizio per la custom skill.

Quando un utente invoca una skill comandandole una determinata funzione, pronuncia: “Alexa, chiedi a [invocation name] di [sample utterance]”. Cioè prima richiama all’attenzione Alexa, poi le fa capire quale skill deve usare e in ultimo le da il comando ben preciso relativo a quell’intent.

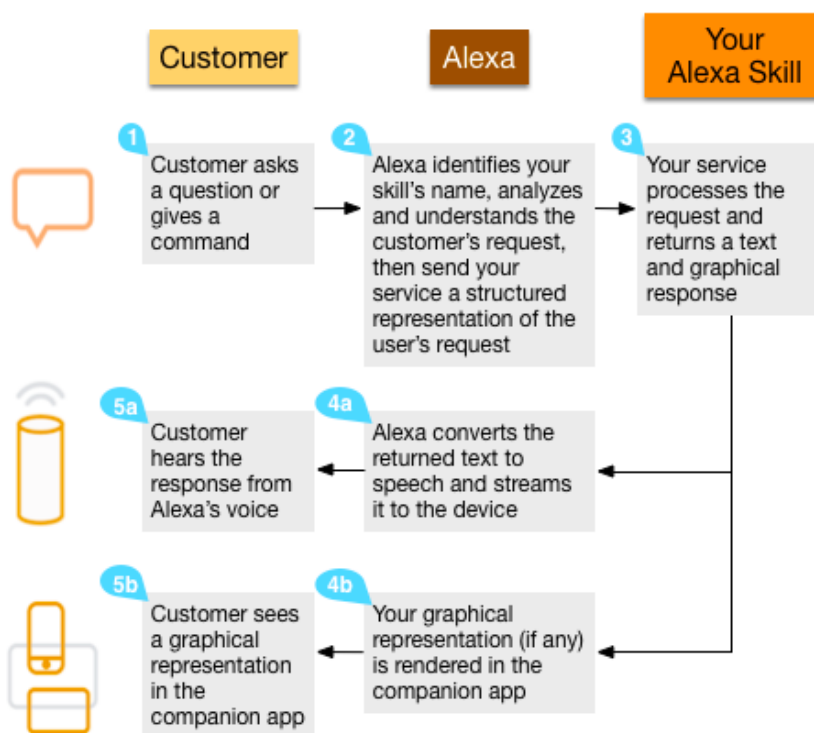


Figura 4 – Architettura dell'interazione vocale con l'Alexa Skill

La skill appena creata può essere testata e poi distribuita. La distribuzione è una pubblicazione che permette di rendere disponibile a tutti gli utenti in rete la propria skill. Se questo non viene fatto, la skill sarà disponibile solo da chi l’ha creata su dispositivi autenticati con il proprio account.

3.2.2 Siri

Siri è l'assistente vocale di Apple. Creato nel 2010 da SRI Inc e acquistato nel 2011 da Apple, Siri è diventato rapidamente parte integrante dell'ecosistema Apple. Funziona su iPhone, iPad, Mac, Apple Watch, Apple TV e HomePod, lo smart speaker dell'azienda.

3.2.3 Google Assistant

Presentato originariamente nel 2016, Google Assistant è stato il successore di Google Now, con il principale miglioramento rappresentato dall'aggiunta di conversazioni bidirezionali. L'Assistente Google fornisce risposte sotto forma di frasi naturali. Google Assistant è disponibile su molti telefoni Android, inclusi gli smartphone Google Pixel, l'altoparlante intelligente Google Home e alcuni altoparlanti di terze parti.

3.2.4 Cortana

A partire dal 2009, Microsoft ha iniziato a distribuire Cortana con tutti i dispositivi Windows 10 e Xbox, ma è disponibile anche per dispositivi mobili Android e Apple. Cortana utilizza il motore di ricerca Bing per rispondere a semplici domande e può impostare promemoria e rispondere ai comandi vocali.

3.2.5 Bixby

Bixby è l'assistente vocale di Samsung, compatibile con smartphone Samsung con Android 7.9 Nougat o superiore. Come Alexa, Bixby risponde ai comandi vocali. Può dare promemoria su attività, controllare la maggior parte delle impostazioni del dispositivo e può eseguire il mirroring dei contenuti dal telefono alla maggior parte delle Smart TV Samsung. Inoltre, insieme alla fotocamera, Bixby può fare acquisti, ottenere una traduzione, leggere i codici QR e identificare una posizione.

Parte II

Progettazione e Implementazione

Capitolo 4

Progetto

In questo capitolo è rappresentata una panoramica della progettazione del sistema IoT. Per prima cosa verranno definiti i requisiti e gli obiettivi principali, poi verrà descritta la struttura del sistema. Verranno illustrate nel dettaglio le caratteristiche, le funzionalità e il ruolo di ogni componente facente parte del sistema.

4.1 Requisiti e obiettivi

Lo scopo del progetto è la realizzazione di un sistema IoT, cioè un sistema volto al controllo vocale dello status di una lampadina. Questo tipo di sistema può essere utilizzato in molti contesti con una vasta gamma di smart objects. Il sistema IoT sviluppato è inoltre in grado di comandare non solo una singola lampadina, ma più dispositivi IoT contemporaneamente purché siano connessi alla stessa rete e siano iscritti allo stesso topic MQTT. L'interazione vocale è gestita dal servizio di Amazon Alexa mediante lo sviluppo di una custom skill sulla Alexa Developer Console. Riprendendo alcuni punti trattati nel capitolo 1, i requisiti fondamentali del sistema IoT sviluppato sono:

- Scalabilità: l'aggiunta o la rimozione dei dispositivi deve essere facile e veloce.

- Eterogeneità: deve essere garantita la possibilità di utilizzo di varie tipologie di device.
- Connessione costante: la connessione a Internet costituisce le fondamenta di un sistema IoT.
- Semplicità di interazione: l'invocazione dei comandi tramite skill deve essere tale da consentire linguaggio discorsivo.

4.2 Architettura

Il controllo vocale del sistema realizzato in questa tesi prevede l'utilizzo e l'interazione di diversi componenti sia hardware che software come mostrato in figura:

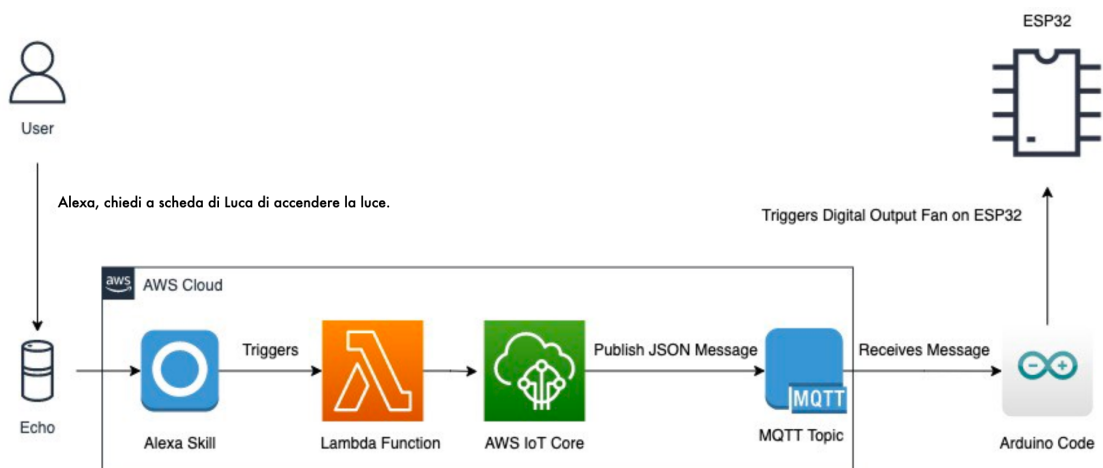


Figura 5 – Architettura Sistema IoT progettato

Di seguito vengono illustrate le varie funzioni e componenti rappresentate all'interno dell'architettura di cui sopra.

4.2.1 Endpoint Alexa (App o Amazon Echo)

L'utilizzo e l'invocazione di funzioni o skill di Alexa è possibile grazie all'interazione vocale con dispositivi quali SmartPhone dotati di app Alexa o dispositivi creati ad hoc da Amazon quali l'Amazon Echo. Questi sono in grado di interagire con l'utente ascoltando le richieste, tradurre ciò che è stato detto in testo tramite l'"*Automatic Speech Recognition (ASR)*" [16] e inviare il tutto all'Alexa Service. Una volta processata la richiesta il device risponderà in maniera discorsiva per avvisare generalmente dell'esito dell'operazione richiesta o per fare ulteriori domande in caso ci sia bisogno di fornire ulteriori input per eseguire ciò che è stato richiesto. La conversazione con Alexa inizia con la sua attivazione tramite la "*wake word*" (di solito questa è "*Alexa*") e, successivamente, si invoca la skill tramite la pronuncia della "*invocation name*". Così l'utente ha accesso a tutte le funzionalità della skill.

4.2.2 Alexa Skill

L'Alexa Skill è interna all'Alexa Service, nel quale avviene la "*Automatic Speech Recognition*", cioè un servizio di traduzione voce - testo sviluppato da Amazon. L'Alexa Service è responsabile dell'interpretazione e instradamento verso l'endpoint della richiesta avanzata dall'utente. Questo dipende dalla tipologia di skill invocata. Tramite l'"*Alexa Skill Kit (ASK)*" è possibile creare nuove skill con nuove funzionalità da implementare con Alexa. A tal proposito, per la creazione di una custom skill è necessario definire un modello di interazione vocale attraverso il quale è possibile

mappare gli input dell'utente agli intenti che l'endpoint può gestire. La costruzione di questo modello di interazione prevede la definizione dei seguenti elementi:

- Intent: rappresenta una funzione principale che viene richiesta dall'utente. Identifica le principali funzionalità della skill. Gli Intent sono definiti in formato JSON e possono essere implementati da frasi dette "Sample Utterances" e parametri detti "Slots".
- Sample Utterance: sono le frasi tipiche che l'utente dovrà pronunciare per invocare un determinato Intent.
- Slot: costituisce una variabile che va ad arricchire l'Intent secondo il suo tipo: numerico, alfabetico, data ecc..

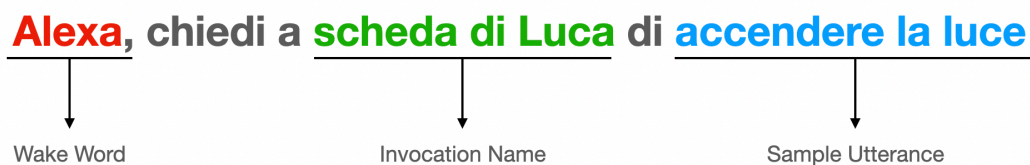


Figura 6 – Invocazione Skill e interazione con i comandi

In figura è mostrato un esempio di come utilizzare la skill per accendere la luce. L'Alexa Service è responsabile della ricezione della risposta da parte dell'endpoint e della sua successiva traduzione testo - voce, per consentire al device in uso (Echo o app Alexa) di fornire un feedback vocale dell'avvenuta accensione della luce.

4.2.3 Lambda Function

AWS Lambda è un servizio di Amazon Web Service per l'esecuzione di codice senza la necessità di fare il provisioning. È dunque possibile eseguire il codice per qualunque tipo di applicazione attraverso un'infrastruttura di calcolo che gestisce autonomamente tutte le risorse. Per ora, il

codice in una funzione Lambda può essere sviluppato in Python, NodeJS, C# e Java. Una volta creata la funzione Lambda può essere caricato il codice precedentemente scritto e completo di tutti i moduli che necessita. Questo servizio comunica con il servizio Alexa tramite un meccanismo di “*request-response*” facendo uso del protocollo HTTP. Quando l’utente interagisce con una skill, la funzione riceve una richiesta in formato JSON che contiene i parametri necessari per fornire una risposta. La logica della skill in questo progetto risiede proprio nella funzione Lambda. Si vedrà nel codice in appendice B che nel file principale sono indicate tutte le condizioni per le quali vale l’invocazione di un Intent piuttosto che di un altro.

4.2.4 AWS IoT Core

AWS IoT Core è un altro servizio di Amazon Web Service presente nell’architettura, che permette di collegare i dispositivi IoT alla piattaforma cloud. Tramite questo servizio è possibile gestire tutti gli smart object collegati ad AWS e le loro policy; inoltre, si può fare un’operazione di test dello scambio di messaggi tramite il protocollo MQTT, facendo una pubblicazione e sottoscrizione ai topic di interesse.

4.2.5 MQTT Topic

MQTT (Message Queing Telemetry Transport) è un protocollo di comunicazione a basso libello e a basso consumo utilizzato nei piccoli dispositivi per trasferire informazioni a basso consumo e bassa capacità di calcolo. Simile all’MQTT è l’HTTP che è nato per il trasferimento di dati di dimensioni più grandi. È stato concepito per i servizi web.

Nell’MQTT ci sono tre elementi principali:

- Publisher: producono informazioni di un certo tipo.
- Subscriber: sono gli “spettatori” che si sottoscrivono a ciò che pubblicano i Publisher.
- Broker: quando ci sono tanti publisher e tanti subscriber

Tutto questo avviene tramite i *topic*, cioè degli argomenti a cui si interessano i subscriber per recepire le informazioni inviate dai publisher.

Nell’architettura proposta, chi fa uso di questo protocollo è chiaramente IoT Core.

4.2.6 Arduino Code

Lo smart object a cui arriva il comando ha bisogno di seguire delle istruzioni per eseguire le richieste che gli arrivano dal Cloud AWS. A tale scopo si utilizza la programmazione in C tramite l’IDE di Arduino per installare lo sketch nello smart object. Il codice di Arduino ha bisogno di due file:

- Header: è un file che aiuta il programmatore nell’utilizzo di librerie durante la programmazione [17]. Al suo interno vengono inseriti i certificati e la rete di connessione.
- Main: è il file principale nel quale risiede la logica dello script che darà indicazioni allo smart object circa le operazioni da eseguire.

4.2.7 ESP32

Si tratta di una serie di microcontrollori a basso costo e bassa potenza

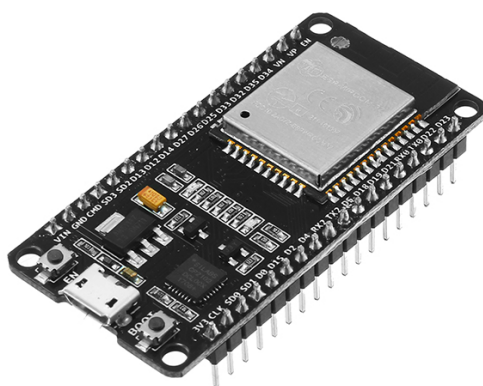


Figura 7 – ESP32

che hanno un modulo Wi-Fi integrato e un modulo Bluetooth dual-band [18] ESP32 è creato e sviluppato da Espressif Systems, una compagnia cinese di Shanghai, ed è il successore del ESP8266.

Il microcontrollore è montato su una board che permette l'interazione con l'ambiente esterno tramite una serie di tasti, porta USB e GPIO. Va alimentata via USB. I due tasti presenti sono uno per il boot e uno per il reset. Il tasto boot in particolar modo viene utilizzando per il download di nuovi script.

Di seguito è rappresentata una panoramica generale delle caratteristiche del microcontrollore in esame.

- CPU: dual-core a 32 bit con due core Xtensa LX6 con Architettura Harvard;
- Clock: fino a 600 DMIPS;
- Memoria interna ed esterna;
- Wi-fi: 2,4 GHz HT40;
- Bluetooth: BLE 4.2 (Bluetooth a bassa energia);
- Periferiche: SPI, I2C, I2S, UART, CAN 2.0 e interfaccia Ethernet;
- ADC: 12 bit (convertitore analogico/digitale);
- Sensori: al tocco, di Hall e temperatura;
- PWM: 1 canale hardware e 16 canali software;
- IO: pin GPIO (ingressi e uscita generici).

Figura 8 – Caratteristiche ESP32

Capitolo 5

Implementazione

La realizzazione del sistema IoT prevede l'integrazione e l'interconnessione di diverse componenti al fine di fornire tutte le funzioni previste e descritte nel capitolo precedente. In questo capitolo viene illustrata passo passo la fase di implementazione del progetto, durante la quale sono state utilizzate diverse componenti software necessarie appunto per favorire l'interazione tra i servizi e gli smart object di cui sopra. Nel dettaglio saranno mostrate l'implementazione della funzione Lambda, la registrazione dello smart object, lo sviluppo della custom skill nell'Alexa Developer Console.

5.1 Flusso di interazione

Il sistema IoT sviluppato prevede l'interazione di diverse componenti che permettono di soddisfare determinate richieste provenienti dall'utente. Una volta associato lo smart object al cloud, esistono due processi principali nel flusso di interazione:

1. Comunicazione tra lo smart object e il Cloud AWS;
2. Interazione tra utente e sistema.

Come già mostrato nel precedente capitolo, l'architettura fa intuire questo flusso di interazione tra le varie componenti del sistema.

Tutto comincia con la connessione del device prima ad Internet, poi al Cloud AWS. A questo punto il sistema è pronto per essere avviato tramite la prima richiesta di invocazione della skill da parte dell'utente. Questo avviene tramite la *wake word* e il nome di invocazione della skill desiderata. Una volta invocata la skill è possibile chiedere ad essa di svolgere una determinata funzione; in entrambe i casi il flusso manda ciò che il device di Amazon ha recepito all'Alexa Service, la quale fa la traduzione voce-testo tramite l'ASR e si avvia verso la funzione Lambda. Nella funzione Lambda risiede il codice in cui c'è la logica della custom skill, e da qui viene invocata proprio la skill tramite il trigger. A questo punto viene inviato il comando al servizio AWS IoT Core che è responsabile della comunicazione diretta con lo smart object. La comunicazione in corso avviene tramite la pubblicazione del payload su un determinato topic tramite il protocollo MQTT in formato JSON. La ricezione del payload da parte dello smart object viene processata attraverso lo sketch di Arduino che ne definisce lo stato del Led (acceso o spento come in questo caso). A metà del flusso, se la richiesta fatta dall'utente viene processata correttamente dall'Alexa Service e dalla funzione Lambda, viene mandato un feedback positivo dal dispositivo di Amazon (Echo o App Alexa) in maniera vocale e discorsiva.

5.2 Registrazione del dispositivo nel Cloud

Per poter gestire il dispositivo (o i dispositivi, a seconda di quanti se ne hanno bisogno), è necessario interfacciarli con il Cloud attraverso il servizio IoT Core. Ogni device è provvisto di un ID univoco e da un nome scelto

dall'utente e ha bisogno di essere associato ad una *policy* per controllare l'accesso al piano dati AWS IoT Core. Il piano dati è costituito da operazioni che consentono la connessione al broker di messaggi, l'invio o la ricezione di messaggi MQTT o aggiornare lo stato di un device. Nella *policy* è fondamentale indicare tre cose:

1. Il nome dello smart object;
2. La regione di utilizzo;
3. Il topic di publish e di subscribe.

Una volta fatto questo si aggiunge lo smart object nella sezione “*Things*” e si scaricano i certificati. Di questi certificati quelli di maggior interesse sono:

1. CA1: certificato di autorità;
2. Device Certificate: certificato del device;
3. Private Key.

5.3 Arduino sketch

Lo sketch di Arduino è lo script scritto in linguaggio C ed installato nella ESP32. Questo racchiude tutta la parte logica relativa alle azioni compiute dalla scheda, provenienti da essa pubblicando sul topic relativo al publish e ricevendo informazioni dal Cloud AWS sottoscrivendosi al topic di subscribe. Nel file principale ci sono varie sezioni, che verranno esaminate di seguito:

- `connectAWS()`: è responsabile della connessione ad Internet e successivamente al Cloud AWS. Invia un messaggio sul monitor seriale dell'avvenuta connessione, uno dei quali proviene dal topic di subscribe proprio perché il feedback viene inviato dal Cloud AWS.

- `publishMessage()`: si occupa della pubblicazione di messaggi sul medesimo topic per inviare dati sul Cloud provenienti dalla scheda.
- `messageHandler()`: è la sezione in cui risiede la logica centrale dell'accensione o spegnimento del LED. Tramite la condizione imposta, a seconda del messaggio che arriva sul topic `Subscribe` la scheda provvederà a cambiare lo stato del LED.
- `setup()`: qui si definiscono gli output e gli input. L'output è rappresentato dal LED della scheda; l'input è invece il tasto boot.
- `loop()`: è l'ultima sezione dove è stata aggiunta la funzione di accensione e spegnimento del LED tramite la pressione del tasto boot. Questo oltre a cambiare lo stato, invierà un messaggio sul topic `Publish` per avvisare dell'avvenuta operazione.

5.4 Alexa custom Skill

Per sviluppare al meglio l'interazione vocale è necessario definire alcuni elementi. Per prima cosa si deve dare un nome alla skill. Successivamente si identificano gli *intent* e le relative *sample utterances*. Come già spiegato nei capitoli precedenti queste tre componenti (nome, intent e sample utterances) sono le tre parti principali che permettono la creazione di una nuova skill. Vista la semplicità di funzionamento del sistema, sono altrettanto basilari gli intents che la skill deve gestire. Allo scopo del progetto sono stati quindi sviluppati due soli intents: uno per spegnere la luce e l'altro per accenderla.

Di seguito sono riportate le sezioni compilate nella skill:

- Nome skill: *SmartLight*
- Nome di invocazione: *scheda di Luca*

- Intent 1: *SwitchOnIntent*
- Intent 2: *SwitchOffIntent*

Per ogni intent sono state inserite alcune frasi tipiche (*sample utterances*) per permettere l'interazione discorsiva per invocare un intent.

SwitchOnIntent:

- Accendere il led
- Accendere la luce
- Luce ON

SwitchOffIntent:

- Spegnere il led
- Spegnere la luce
- Luce OFF

L'interazione avviene completamente in lingua italiana, perché così è stato convenientemente deciso.

Una volta terminata la fase di compilazione dei vari intents, nella sezione "Interaction Model" compare una versione aggiornata che descrive gli intents in formato JSON.

L'utente, in fase di interazione vocale con la skill, potrebbe non pronunciare il comando per intero, potrebbe non pronunciare bene le sue intenzioni o potrebbe soltanto invocare la skill tramite il suo nome. Per ognuno di questi casi c'è una risposta da parte di Alexa che permette all'utente di avere un feedback ben preciso. Di seguito alcuni casi:

Utente: “Alexa, avvia la scheda di Luca.”

Alexa: “Ciao. Come posso aiutarti?”

Utente: “Alexa, chiedi a scheda di Luca di accendere la luce.”

Alexa: “Luce accesa.”

Utente: “Alexa, chiedi a scheda di Luca di [parole incomprensibili].”

Alexa: “Spiacente.”

Questi sono i tre feedback possibili in tre scenari diversi: richiesta di avvio, richiesta di un’azione, richiesta incomprensibile

5.5 Funzione Lambda

La funzione Lambda è il codice che viene eseguito dal medesimo servizio di AWS per processare le richieste effettuate dall’utente nel momento in cui inizia la sua interazione con l’Alexa Custom Skill. Il codice sviluppato è stato scritto in NodeJS. Esso è composto principalmente dal *Lambda Handler*, *Intent Handlers* e *Response Builder*.

5.5.1 Lambda Handler

Quando si crea la funzione, la prima cosa da fare dopo aver scelto il linguaggio del codice e la sua policy, è impostare un *handler*. Questa è una funzione interna al codice che può essere invocata da AWS Lambda quando il servizio riceve una richiesta da parte di Alexa. Le richieste di

questo genere sono inviate in formato JSON e presentano una struttura del genere:

```
"request": {
  "type": "IntentRequest",
  "requestId": "id",
  "locale": "it-IT",
  "timestamp": "2022-01-21T16:48:24Z",
  "intent": {
    "name": "SwitchOnIntent",
    "confirmationStatus": "NONE"
  }
}
```

In questo caso è stata avanzata una richiesta da parte dell'utente di accendere la luce, e il Lambda Handler è responsabile di verificare che questa richiesta sia corretta. Successivamente invoca il relativo Intent Handler.

5.5.2 Intent Handler

Questo Handler si occupa direttamente della gestione delle richieste provenienti dall'utente. Questo interagisce con gli intent presenti nella skill e, ne esiste uno per ogni intent precedentemente creato. La struttura di un Intent Handler è sempre la stessa, anche se gli intents creati nella custom skill possono essere di tipo diverso.

Un esempio di Intent Handler per lo scopo di questo progetto è il seguente:

```

'SwitchOnIntent': function () {
    var params = {
        topic: topic,
        payload: "on",
        qos:0
    };
    iotData.publish(params, (error, data)=>{
        if (!error){this.emit(':tell', 'Luce accesa.')}
        }else{this.emit(':tell', 'Qualcosa è andato storto.')}
    });
},

```

Questo Intent Handler è il responsabile dello *SwitchOnIntent* precedentemente creato nella custom skill. Come si può notare al suo interno vengono identificati il topic (riferito al subscribe indicato nel codice), la condizione del payload relativa a questo Handler e il Quality of Service. Se tutto va a buon fine viene inviato un feedback vocale che avvisa dell'avvenuta accensione della luce, altrimenti dice che qualcosa è andato storto.

5.5.3 Response Builder

Il *Response Builder* si occupa di inviare una risposta sempre in formato JSON ad Alexa. La struttura di una risposta è la seguente:

```

"response": {
    "outputSpeech": {

```

```
        "type": "SSML",
        "ssml":    "<say>    Luce    accesa.
</say>"
    },
    "shouldEndSession": true,
    "type": "_DEFAULT_RESPONSE"
},
```

Il campo *outputSpeech* contiene il testo che verrà tradotto in voce dall'Alexa Service per fornire il feedback vocale al nostro comando. Nel campo *shouldEndSession* invece è indicato alla skill se deve chiudere la sessione subito dopo l'avvenuta richiesta o meno. In questo caso viene chiuso tutto.

Parte III

Risultati

Capitolo 6

Risultati

In questo capitolo verranno analizzati i risultati provenienti dal sistema IoT sviluppato con il relativo funzionamento in base alle richieste lato utente. Inoltre, verranno osservati i vari valori di latenza del servizio AWS IoT Core e del relativo Automatic Speech Recognition per la gestione della traduzione voce - testo quando si avanza una richiesta e della successiva traduzione testo - voce quando l'Alexa Service fornisce un feedback vocale all'utente.

6.1 Ambito di funzionamento

Come descritto nei capitoli precedenti, il sistema IoT sviluppato prevede il comando vocale tramite l'Alexa service dell'accensione e spegnimento del led sulla scheda ESP32, la quale si interfaccia con il cloud AWS tramite l'AWS IoT Core. La logica del funzionamento complessivo del sistema risiede nel codice caricato nella funzione Lambda scritto in NodeJS il cui utilizzo quotidiano non ha bisogno di essere alimentato a batteria: trattandosi dell'accensione e dello spegnimento di una luce è plausibile che nella maggior parte dei casi questa sia alimentata dalla rete elettrica domestica.

6.2 Configurazione

Considerato il contesto applicativo del sistema IoT sviluppato, il test di funzionamento è stato effettuato utilizzando una scheda ESP32 dotata di un modulo Wi-Fi, tramite il quale è possibile il collegamento alla rete, al servizio AWS e quindi lo scambio di informazioni bidirezionale. Questo scambio di informazioni è possibile grazie all'utilizzo del protocollo MQTT. Lo sketch di Arduino caricato sulla scheda è lo stesso descritto nel paragrafo 5.3. Come output è stato utilizzato il led presente sulla board e corrispondente al pin 0; inoltre per simulare l'accensione e lo spegnimento da interruttore manuale è stato utilizzato il pulsante boot della scheda che ne sostituisce il compito. Riguardo quest'ultimo, per rendere coerente la natura dell'intero sistema, ogni volta che viene premuto viene inviato (pubblicato sul topic publish) un messaggio che avvisa dell'avvenuta accensione del led. Quindi, chiunque sia iscritto a quel topic, viene avvisato di tale azione.

Come già accennato nella sezione precedente, il sistema non richiede necessariamente (nella maggior parte dei casi di utilizzo quotidiano) un tipo di alimentazione autonoma, quindi è stato considerato alimentato tramite la rete e a tale scopo, in fase di test, l'alimentazione è stata data via USB. Per eseguire il test è necessario aver fatto il login nel proprio account di Amazon per avere accesso alla propria custom skill, che è disponibile solo in ogni dispositivo con accesso allo stesso account finché non viene distribuita e pubblicata. Il comando vocale può essere dato da App sul proprio smartphone o dall'Amazon Echo.

6.3 ASR Delay

Inizialmente, l'ASR di Alexa veniva eseguito nel Cloud [20] e questo provocava dei ritardi dovuti diversi a diversi fattori, tra cui la connessione e il traffico di informazioni nella rete. Per questo motivo si è deciso di spostare l'ASR direttamente nei device abilitati al servizio Alexa, quindi App o dispositivi Amazon Echo, favorendo così una velocizzazione della traduzione voce - testo e viceversa.

6.4 Trip Delay

Considerando il caso più semplice in cui ci sono solo un Client Publisher P ed un Client Subscriber S, si vuole analizzare nel dettaglio tutto ciò che avviene nel mezzo della comunicazione, specialmente i tempi di impiego del percorso, dal momento in cui viene avanzata la richiesta da P al momento in cui viene ricevuta da S. Tutto questo è possibile tenendo conto degli istanti di tempo in cui avvengono queste azioni:

1. P pubblica un messaggio con QoS pari a 0 all'istante t_0 ;
2. Il Broker riceve il messaggio all'istante t_1 ;
3. Il Broker invia un ack a P nello stesso istante, e viene recapitato all'istante t_4 ;
4. Il Broker inoltra il messaggio a S all'istante t_2 ;
5. S riceve il messaggio all'istante t_3 ;
6. S invia un ack al Broker nello stesso istante, e viene recapitato all'istante t_2 .

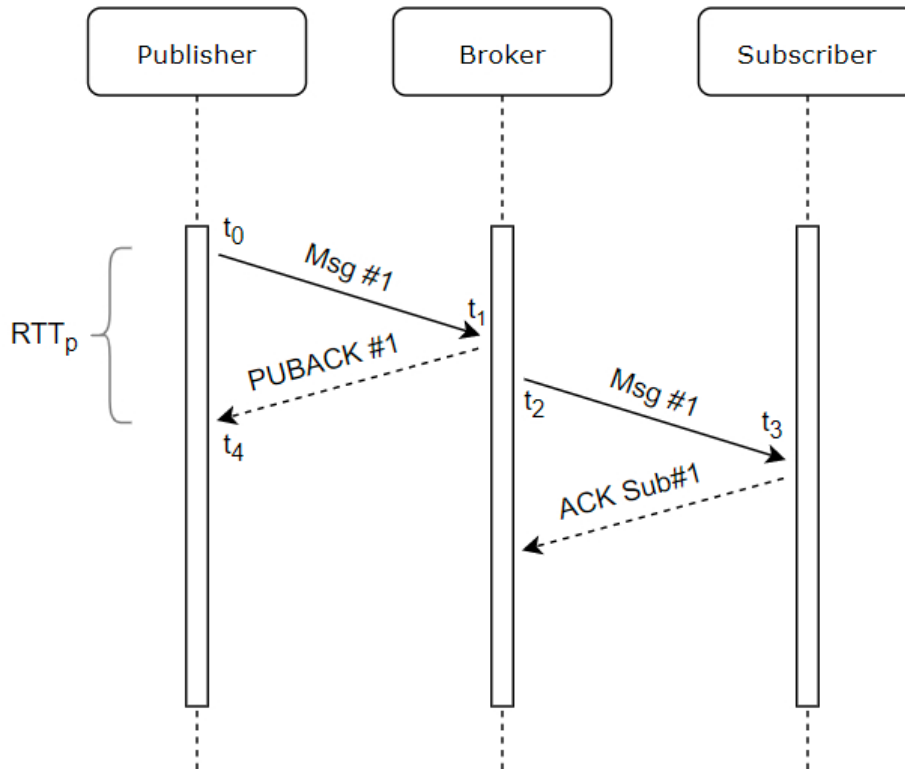


Figura 9 – Flusso di interazione tra Publisher, Broker e Subscriber

Nella figura è illustrato il percorso che segue il messaggio con i relativi feedback che vengono inviati ogni volta che il messaggio arriva ad uno degli elementi della rete. Il Broker Service Time, cioè il tempo che impiega il Broker a compiere le sue azioni è pari a:

$$T_c = t_2 - t_1 \quad (6.1)$$

Questo valore è possibile consultarlo nel sistema di log della piattaforma Cloud AWS. Partendo dagli algoritmi del protocollo NTP è possibile calcolare il Round Trip Delay (RTD):

$$RTD = (t_3 - t_0) - (t_2 - t_1) \quad (6.2)$$

Inoltre, si può prendere in considerazione il Round Trip Time lato P (RTTp):

$$RTT_p = t_4 - t_0 \quad (6.3)$$

Ci si aspetta che il valore del Broker Service Time sia uguale alla differenza tra l'istante t_3 (arrivo del messaggio al client S) e l'istante t_4 (arrivo dell'ack al client P):

$$T_C = t_2 - t_1 = t_3 - t_4 \quad (6.4)$$

Questo è assunto sulla base della supposizione che tutti i ritardi siano simmetrici.

6.5 Risultati

L'impostazione e la configurazione del sistema IoT sviluppato ha dato modo di testare in svariati modi (per quanto la natura del sistema stesso possa permetterlo) il suo funzionamento. Come già accennato sopra, per avviare il sistema, è sufficiente avere la skill non necessariamente distribuita: basta avere l'accesso al proprio account Amazon sui dispositivi che si andranno ad utilizzare per inviare il comando vocale all'Alexa Service. Per prima cosa si alimenta la scheda via USB e si attende la sua connessione prima alla rete Internet poi al Cloud AWS. Questo può essere verificato nel monitor seriale dell'Arduino IDE, in cui ci sarà un log di tutto ciò che coinvolge la ESP32.

```
18:56:39.639 -> ets Jun 8 2016 00:22:57
18:56:39.639 ->
18:56:39.639 -> rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
18:56:39.639 -> configsip: 0, SPIWP:0xee
18:56:39.639 -> clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
18:56:39.639 -> mode:DIO, clock div:1
18:56:39.639 -> load:0x3fff0018,len:4
18:56:39.639 -> load:0x3fff001c,len:1216
18:56:39.679 -> ho 0 tail 12 room 4
18:56:39.679 -> load:0x40078000,len:10944
18:56:39.679 -> load:0x40080400,len:6388
18:56:39.679 -> entry 0x400806b4
18:56:40.043 -> Connecting to Wi-Fi
18:56:40.556 -> .Connecting to AWS IOTAWS IoT Connected!
```

Figura 10.- Verifica connessione da monitor seriale

Nell'immagine è indicato il *timestamp* di ogni evento, seguito dalle informazioni di maggior interesse tra cui, appunto, la voce che avvisa l'avvenuta connessione alla rete Internet e l'avvenuta connessione al Cloud AWS. In questo modo c'è la sicurezza di avere una corrispondenza tra la scheda (che rappresenta lo smart object) e il servizio AWS IoT Core.

Per avere un'ulteriore prova di questo collegamento ci si può spostare nella sezione *MQTT Test Client* del servizio IoT Core e verificare se effettivamente scheda e Cloud riescono a conversare. A tale scopo si pubblica un messaggio sia dal Device sia dal Cloud.

Lo sketch installato nella ESP32 prevede l'accensione e lo spegnimento della luce tramite il pulsante *boot*, simulando un vero e proprio interruttore; a questa azione corrisponde la pubblicazione di un messaggio di avviso sul topic *esp32/pub*:

```
19:13:52.449 -> LED ON
19:13:52.449 -> number of button pushes: 3
19:13:52.593 -> Published Message:{"Action":"pressed"}
19:13:52.593 -> =====
```

Figura 12 – Pubblicazione avviso di azionamento dell'interruttore

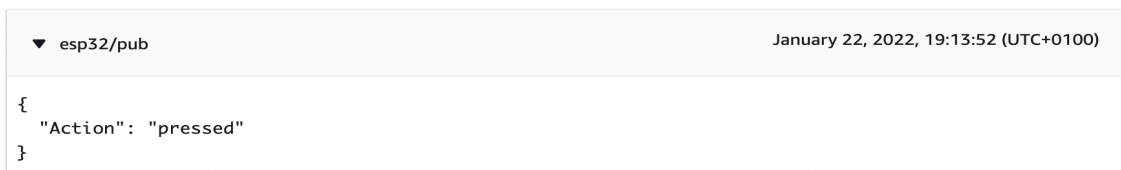


Figura 11 – Feedback dell'avvenuta accensione della luce tramite pulsante

Come si evince dalle immagini, l'accensione manuale del LED produce un messaggio di avviso i cui istanti coincidono.

Per completare il test di questa prima fase, è sufficiente comandare lato Cloud lo stato desiderato della luce.

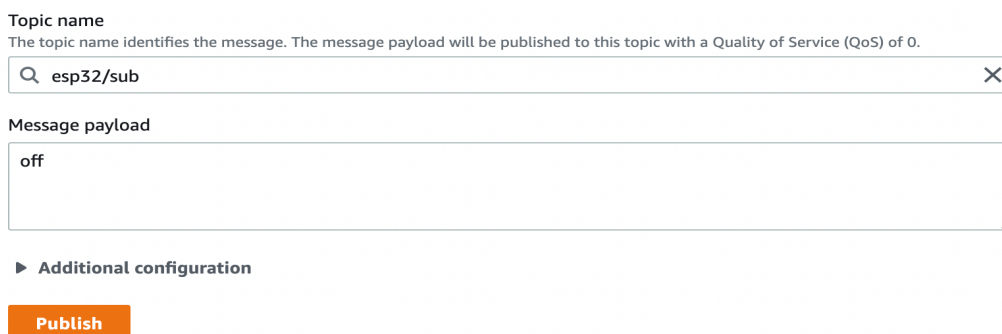


Figura 13 – Spegnimento luce lato Cloud

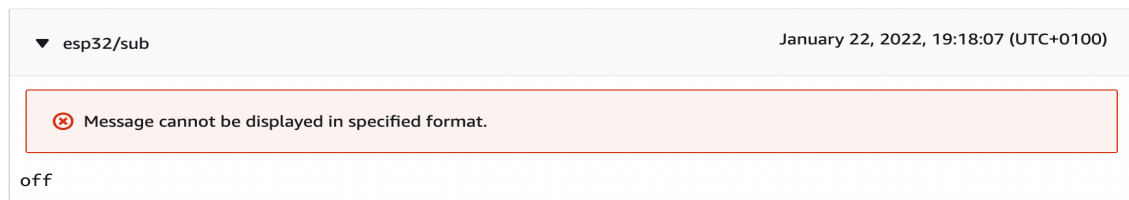


Figura 15 – Feedback dell'azione dal tester MQTT

```
19:18:07.620 -> AWS incoming MQTT topic: esp32/sub
19:18:07.620 ->
19:18:07.620 -> AWS incoming MQTT payload: off
```

Figura 14 – Feedback dell'azione dal monitor seriale

Anche in questo caso gli orari coincidono.

Tutto quello che si è fatto ora, deve essere comandato tramite l'interazione vocale con Alexa. La custom skill sviluppata è stata denominata *Smart-Light* e per testarla è sufficiente aprire il tab *Test* dell'Alexa Developer Console e digitare testualmente il comando o pronunciarlo vocalmente.

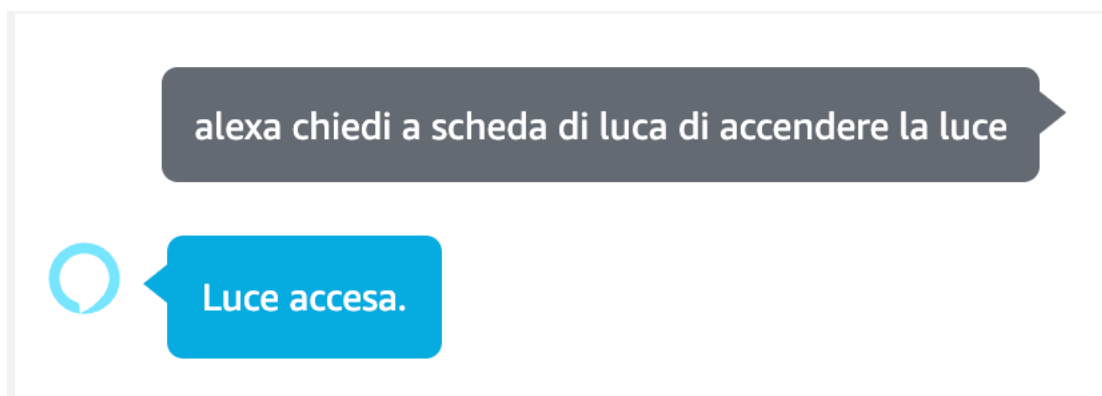


Figura 16 – Interazione vocale con Alexa tramite il tester dell'Alexa Developer Console

Queste operazioni producono dei file JSON che vengono scambiati con la funzione Lambda. In particolare, successivamente all'invio del comando vocale ad Alexa, si apriranno due schermate: una di input e una di output.

```
JSON Input 1
1- {
2-   "version": "1.0",
3-   "session": {
4-     "new": true,
5-     "sessionId": "amzn1.echo-api.session.76ed16d
6-     "application": {
7-       "applicationId": "amzn1.ask.skill.a3b4f0
8-     },
9-     "attributes": {},
10-    "user": {
11-      "userId": "amzn1.ask.account.AEOKFCV5AKT
12-      "permissions": {
13-        "consentToken": "eyJ0eXAiOiJKV1QiLCJ
14-      }
15-    }
16-  },
17-  "context": {
18-    "Viewports": [
19-      {
20-        "type": "APL",
21-        "id": "main",
22-        "shape": "RECTANGLE",
23-        "dpi": 213,
24-        "presentationType": "STANDARD",
25-        "canRotate": false,
26-        "configuration": {
27-          "current": {
28-            "mode": "HUB",
29-            "video": {
30-              "source": [

JSON Output 1
1- {
2-   "body": {
3-     "version": "1.0",
4-     "response": {
5-       "outputSpeech": {
6-         "type": "SSML",
7-         "ssml": "<say> Luce accesa. </say>"
8-       },
9-       "shouldEndSession": true,
10-      "type": "_DEFAULT_RESPONSE"
11-    },
12-    "sessionAttributes": {},
13-    "userAgent": "ask-nodejs/1.0.25 Node/v14.18.1"
14-  }
15- }
```

Figura 17 – Input e Output in formato JSON generate da Alexa

La schermata di input è relativa al *Lambda Handler* e la schermata di output è relativa al *Risponde Builder*. Il JSON input può essere copiato e incollato direttamente nel test della funzione Lambda per verificarne il corretto funzionamento.

Analisi dei ritardi

Grazie allo spostamento dell'Automatic Speech Recognition dal Cloud agli endpoint è possibile notare in fase di richiesta una traduzione voce - testo praticamente immediata. Studi condotti in [21] affermano che l'utilizzo simultaneo di più *encoder* riducono i costi computazionali di una rete neurale alla base dell'Automatic Speech Recognition più del 45% senza inficiare sull'accuratezza dei testi tradotti. Questo significa avere una latenza pari a *29ms* del tempo che intercorre tra la nostra voce e la traduzione in testo da parte del servizio.

L'altro tipo di latenza in esame come già accennato è quella riguardante il tempo che intercorre tra l'invio del messaggio lato Publisher e la ricezione dello stesso lato Subscriber. Questa latenza è stata denominata *Trip Delay* ed è stata calcolata osservando la differenza dei tempi di partenza e di arrivo dei messaggi tramite i log presenti sul servizio AWS CloudWatch. I log in esame hanno il seguente formato:

```
{“timestamp”: “2022-01-22 19:18:07.620”,
“logLevel”: “INFO”,
“traceId”: “xxxxxxxx-xxxx-xxxx-xxxx-xx”,
“accountId”: “0123456789”,
“status”: “Success”,
“eventType”: “Publish-In”,
“protocol”: “MQTT”,
“topicName”: “esp32/sub”,
“clientId”: “cliendId”,
“principalId”: “principalIdentification”,
“sourceIp”: “xxx.xxx.xxx.xxx”,
“sourcePort”: 01234 }
```

Nel campo *eventType* è indicato il valore “Publish-In”. Questo sta a significare che un messaggio è arrivato al broker. Nel caso in cui invece ci fosse stato “Publish-Out”, avrebbe indicato l'invio del messaggio da parte del broker verso il subscriber. Il valore a cui prestare attenzione al fine di calcolare il tempo che impiega il messaggio ad arrivare a destinazione, è il *timestamp*. È necessario quindi fare la differenza tra i due valori di timestamp presenti nei file JSON prodotti dal CloudWatch.

Su un campione di otto trasmissioni effettuate sono stati calcolati la differenza dei timestamp e la deviazione standard dei messaggi inviati dal Publisher al Subscriber passando per il Broker.

I risultati sono riportati in tabella e mostrano in linea di massima un andamento coerente per ogni trasmissione effettuata, che si aggira mediamente sui 27ms.

Trasmissioni	Durata (ms)	Dev. Standard
#1	26.1703	0.1974
#2	29.042	3.5295
#3	27.4792	1.2097
#4	27.5687	4.2957
#5	27.2277	6.1284
#6	27.1459	5.7324
#7	27.0698	5.0916
#8	27.0938	4.4566

Tabella 1 – Tempi e deviazione standard di AWS basati sui log del Cloudwatch

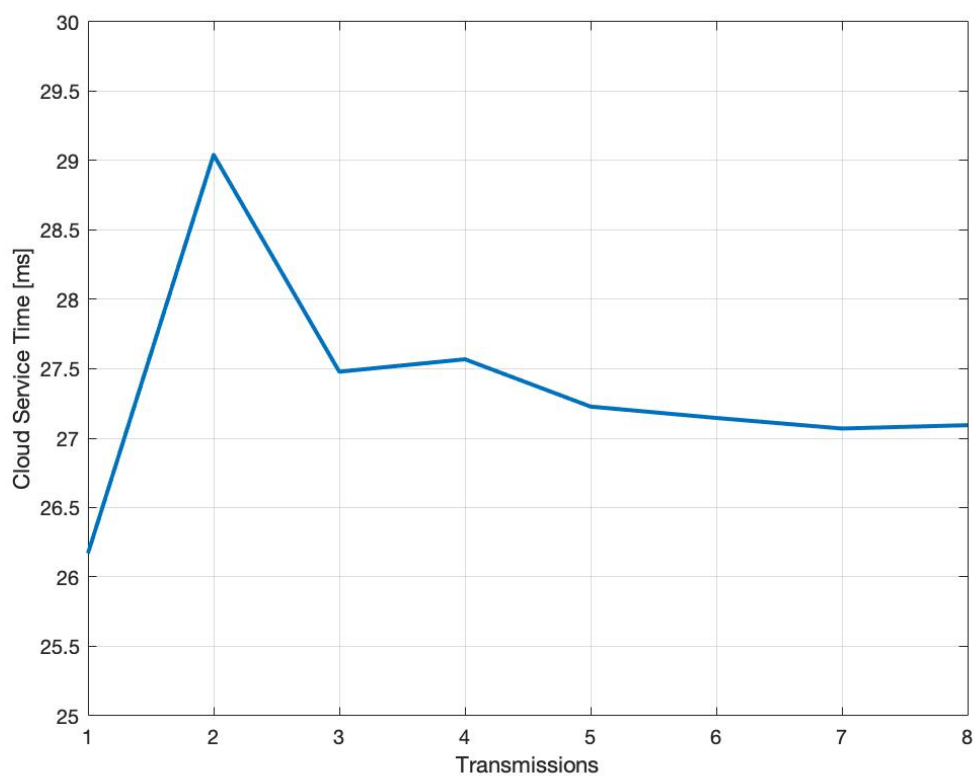


Figura 18 - Ritardi di comunicazione

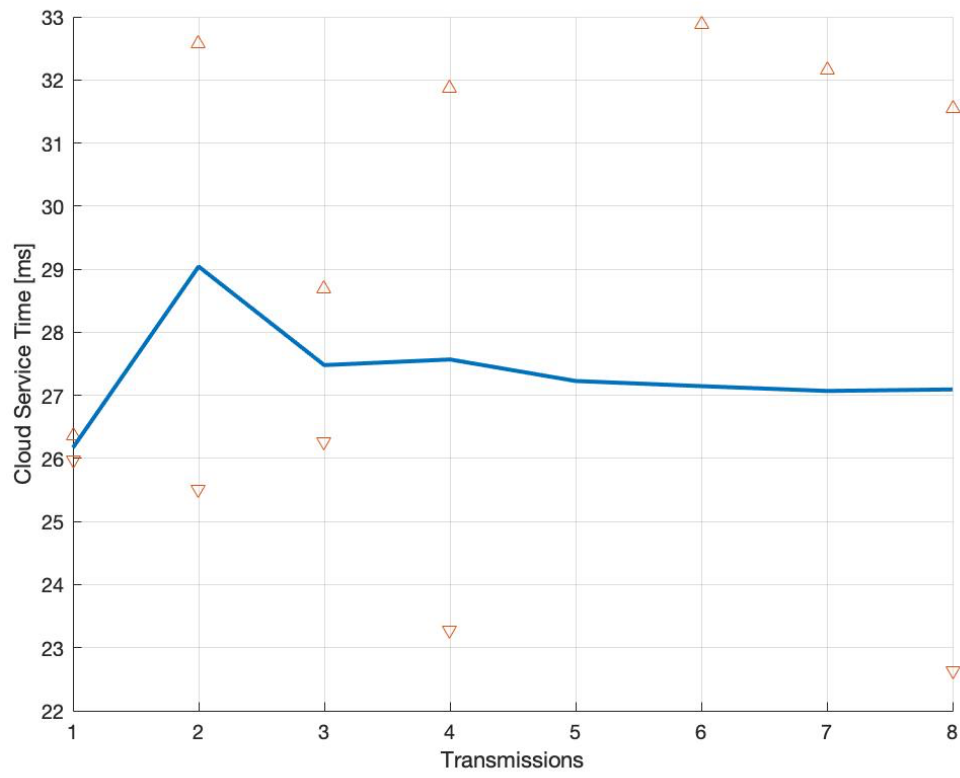


Figura 19 - Round Trip Delay con scarto quadratico medio

Conclusioni

In questa tesi è stato descritto il sistema IoT sviluppato per la gestione vocale di dispositivi IoT mediante l'utilizzo di una Alexa custom Skill denominata "*SmartLight*". Quindi attraverso l'interazione vocale con Alexa, l'utente è in grado di inviare comandi agli smart objects collegati nel Cloud AWS relazionandosi direttamente con il servizio AWS IoT Core; inoltre offre la possibilità di avvisare l'utente sui propri device dell'avvenuta commutazione dello smart object tramite switch fisico (come potrebbe essere un interruttore della luce). La caratteristica principale di questo sistema è quella di cambiare lo stato della luce portandola da spenta a accesa (e viceversa) tramite la comodità della sola interazione vocale con l'assistente di Amazon. Il suo utilizzo può essere esteso a molteplici scenari, viste le possibilità che offre questo tipo di sistema, come sistemi SmartHome. Data l'importanza in alcuni ambiti della tempestività delle azioni da comandare ai vari endpoint, sono stati studiati i tempi di comunicazione tramite il calcolo dei ritardi del Cloud AWS compreso del servizio di Speech Recognition di Alexa. Preso un campione di otto comunicazioni effettuate, sono stati studiati i vari log in uscita dai processi per valutarne i trip Delay e lo scarto quadratico medio.

Il sistema *SmartLight* può essere installato in una rete domestica e può essere implementato con le seguenti migliorie:

- Aggiungere collegamenti simultanei a molteplici device sparsi in diverse stanze in modo da poterne controllare più di uno soltanto: la skill di Alexa verrebbe quindi arricchita di nuovi intents e slot che ne identifica la posizione e altri parametri dei vari device;

- Aggiungere funzionalità di controllo della temperatura della luce e della sua intensità;
- Sviluppare un sistema di invio del feedback dai device sul loro status ogni qualvolta viene richiesto dall'utente e feedback automatici in caso di guasto di uno di essi.

Appendice A

Arduino Sketch

Di seguito verrà implementato il codice in C dei due file che compongono lo sketch installato nella ESP32.

A.1 Secrets.h

```
#include <pgmspace.h>

#define SECRET
#define THINGNAME "ESP32"

const char WIFI_SSID[] = "xxx";
const char WIFI_PASSWORD[] = "xxx";
const char AWS_IOT_ENDPOINT[] = "xxx";

// Amazon Root CA 1
static const char AWS_CERT_CA[] PROGMEM = R"EOF(
-----BEGIN CERTIFICATE-----

-----END CERTIFICATE-----
)EOF";

// Device Certificate
static const char AWS_CERT_CRT[] PROGMEM = R"KEY(
-----BEGIN CERTIFICATE-----"
```

```

-----END CERTIFICATE-----
)KEY";

// Device Private Key
static const char AWS_CERT_PRIVATE[] PROGMEM = R"KEY(
-----BEGIN RSA PRIVATE KEY-----

-----END RSA PRIVATE KEY-----
)KEY";

```

A.2 main.ino

```

#include "secrets.h"
#include <WiFiClientSecure.h>
#include <MQTTClient.h>
#include <ArduinoJson.h>
#include "WiFi.h"

// The MQTT topics that this device should publish/sub-
scribe
#define AWS_IOT_PUBLISH_TOPIC    "esp32/pub"
#define AWS_IOT_SUBSCRIBE_TOPIC "esp32/sub"

const int buttonPin = 0;
const int ledPin = 2;

int buttonPushCounter = 0;
int buttonState = 0;
int lastButtonState = 0;
int led = 0;

```

```

WiFiClientSecure net = WiFiClientSecure();
MQTTClient client = MQTTClient(256);

void connectAWS()
{
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

  Serial.println("Connecting to Wi-Fi");

  while (WiFi.status() != WL_CONNECTED){
    delay(500);
    Serial.print(".");
  }

  // Configure WiFiClientSecure to use the AWS IoT de-
  vice credentials
  net.setCACert(AWS_CERT_CA);
  net.setCertificate(AWS_CERT_CRT);
  net.setPrivateKey(AWS_CERT_PRIVATE);

  // Connect to the MQTT broker on the AWS endpoint we
  defined earlier
  client.begin(AWS_IOT_ENDPOINT, 8883, net);

  // Create a message handler
  client.onMessage(messageHandler);

  Serial.print("Connecting to AWS IOT");

```



```

while (!client.connect(THINGNAME)) {
    Serial.print(".");
    delay(100);
}

if(!client.connected()){
    Serial.println("AWS IoT Timeout!");
    return;
}

// Subscribe to a topic
client.subscribe(AWS_IOT_SUBSCRIBE_TOPIC);

Serial.println("AWS IoT Connected!");
}

void publishMessage()
{
    char payload[512];
    sprintf(payload, "{\"Action\": \"pressed\"}");
    client.publish(AWS_IOT_PUBLISH_TOPIC, payload);
    Serial.print("Published Message:");
    Serial.println(payload);
}

void messageHandler(String &topic, String &payload) {
    Serial.println("\nAWS incoming MQTT topic: " +
topic);
}

```

```

    Serial.println("\nAWS incoming MQTT payload: " + payload);

    if (payload == "on")    digitalWrite(ledPin, HIGH);
    if (payload == "off")   digitalWrite(ledPin, LOW);
}

void setup() {
    Serial.begin(115200);
    pinMode(ledPin, OUTPUT);
    pinMode(buttonPin, INPUT);

    connectAWS();
}

void loop() {
    buttonState = digitalRead(buttonPin);

    if (buttonState != HIGH) {
        buttonPushCounter++;
        if ((buttonPushCounter%2) == 0) {
            digitalWrite(ledPin, LOW);
            Serial.println("LED OFF");
            Serial.print("number of button pushes: ");
            Serial.println(buttonPushCounter);
            publishMessage();
            Serial.println("=====");
        }else
        if ((buttonPushCounter%2) != 0){
            digitalWrite(ledPin, HIGH);

```

```
    Serial.println("LED ON");
    Serial.print("number of button pushes: ");
    Serial.println(buttonPushCounter);
    publishMessage();
    Serial.println("=====");
}
// Delay a little bit to avoid bouncing
delay(1258);
}
client.loop();
}
```

Appendice B

Lambda Function

In questa sezione verrà mostrata la logica sviluppata in NodeJS riguardo la funzione Lambda. Qui risiede il corpo centrale del sistema IoT.

B.1 index.js

```
var APP_ID = "xxx"; // Alexa skill ID
var AWS = require('aws-sdk');
var Alexa = require("alexa-sdk");

AWS.config.region = "eu-west-1";
var iotData = new AWS.IotData({endpoint: "xxx"});
var topic = "esp32/sub";

exports.handler = function(event, context, callback) {
    var alexa = Alexa.handler(event, context);
    alexa.appId = APP_ID;
    alexa.registerHandlers(handlers);
    alexa.execute();
};

var handlers = {
```

```

    'LaunchRequest': function () {
        this.emit(':tell', 'Ciao. Come posso aiu-
tarti?');
    },

    'SwitchOnIntent': function () {
        var params = {
            topic: topic,
            payload: "on",
            qos:0
        };
        iotData.publish(params, (error, data)=>{
            if (!error){this.emit(':tell', 'Luce ac-
cesa.')}

            }else{this.emit(':tell', 'Qualcosa è andato
storto.')}}
        });

    },

    'SwitchOffIntent': function () {
        var params = {
            topic: topic,
            payload: "off",
            qos:0
        };
        iotData.publish(params, (error, data)=>{
            if (!error){this.emit(':tell', 'Luce
spenta.')}

```

```
        }else{this.emit(':tell', 'Qualcosa è andato
storto.')}
    });

},
'Unhandled': function () {
    this.emit(':tell', 'Spiacente.')}
};
```

Appendice C

Alexa Custom Skill

In quest'ultima sezione è indicato un resoconto della struttura in JSON della skill sviluppata.

C.1 SmartLight

```
{
  "interactionModel": {
    "languageModel": {
      "invocationName": "scheda di luca",
      "intents": [
        {
          "name": "AMAZON.CancelIntent",
          "samples": []
        },
        {
          "name": "AMAZON.HelpIntent",
          "samples": []
        },
        {
          "name": "AMAZON.StopIntent",
          "samples": []
        },
        {
```

```

        "name": "AMAZON.NavigateHomeIntent",
        "samples": [],
    },
    {
        "name": "SwitchOnIntent",
        "slots": [],
        "samples": [
            "accendere il led",
            "luce on",
            "accendere la luce"
        ]
    },
    {
        "name": "SwitchOffIntent",
        "slots": [],
        "samples": [
            "spegnere il led",
            "luce off",
            "spegnere la luce"
        ]
    }
],
"types": []
}
}
}

```


Bibliografia

- [1] C. Petrov, «techjury.net,» Gennaio 2022. [Online]. Available: <https://techjury.net/blog/internet-of-things-statistics/#gref>.
- [2] «Guida all'Internet of Things,» [Online]. Available: https://blog.osservatori.net/it_it/cos-e-internet-of-things#:~:text=Cosa%20si%20intende%20per%20Internet%20delle%20Cose,Auto%20DID%20Center%20di%20Massachussetts..
- [3] IEEE, «Special Report: The Internet of Things,» 2014.
- [4] A. Tumino, «Internet of Things: gli oggetti intelligenti prima di ogni "cosa",» 24 Aprile 2019. [Online]. Available: https://blog.osservatori.net/it_it/internet-of-things-oggetti-intelligenti-prima-di-ogni-cosa?hsLang=it-it.
- [5] E. Borgia, «The Internet of Things vision: Key features, applications and open issues,» 2014.
- [6] NIST, «The NIST definition of cloud computing,» 2011.
- [7] A. B. L. P. R. C. Paola Pierleoni, *Amazon, Google and Microsoft Solutions for IoT: Architectures and a Performance Comparison*, 2020.
- [8] «Amazon Web Services,» [Online]. Available: https://it.wikipedia.org/wiki/Amazon_Web_Services.
- [9] «What is AWS,» [Online]. Available: https://aws.amazon.com/it/what-is-aws/?nc1=f_cc.
- [10] «AWS IoT Core,» [Online]. Available: <https://aws.amazon.com/it/iot-core/>.
- [11] «Cos'è AWS Lambda?,» [Online]. Available: https://docs.aws.amazon.com/it_it/lambda/latest/dg/welcome.html.
- [12] «AWS Identity and Access Management (IAM),» [Online]. Available: <https://aws.amazon.com/it/iam/>.
- [13] «Assistente virtuale,» [Online]. Available: https://it.wikipedia.org/wiki/Assistente_virtuale.
- [14] A. Turing, «COMPUTING MACHINERY AND INTELLIGENCE,» 1950.
- [15] «wearable-home,» [Online]. Available: <https://www.wearable-home.com/assistente-vocale/#anchor1>.
- [16] «Understand Custom Skills,» [Online]. Available: <https://developer.amazon.com/en-US/docs/alexa/custom-skills/understanding-custom-skills.html>.
- [17] «What Is Automatic Speech Recognition?,» [Online]. Available: <https://developer.amazon.com/it-IT/alexa/alexa-skills-kit/asr>.

- [18] «Header file,» [Online]. Available: https://it.wikipedia.org/wiki/Header_file.
- [19] «ESP32,» [Online]. Available: <https://en.wikipedia.org/wiki/ESP32>.
- [20] G. P. Strimel, «How to make on-device speech recognition practical,» 2 Settembre 2021. [Online]. Available: <https://www.amazon.science/blog/how-to-make-on-device-speech-recognition-practical>.
- [21] G. P. S. J. S. A. R. Jonathan Macoskey, «Amortized neural networks for low-latency speech recognition,» 2021. [Online]. Available: <https://www.amazon.science/publications/amortized-neural-networks-for-low-latency-speech-recognition>.