UNIVERSITÀ POLITECNICA DELLE MARCHE

FACOLTÀ DI INGEGNERIA

Corso di Laurea Triennale in
Ingegneria Informatica e dell'Automazione

Progettazione e sviluppo di un servizio cloud basato su serverless computing per la predizione di serie temporali nell'ambito del progetto ReCity

Design and development of a cloud service based on serverless computing for time series prediction within the ReCity project

Relatore:
Prof. Mancini Adriano

Laureando:
De Ritis Riccardo

Anno Accademico 2022-2023

Indice

1 Introduzione 4
1.1 Obiettivi 6
1.2 Struttura della Tesi 7

2 Cloud Computing 8
2.1 Modelli di distribuzione dei servizi 10
2.1.1 Infrastructure as a Service (IaaS) 11
2.1.2 Platform as a Service (PaaS) 11
2.1.3 Software as a Service (SaaS) 12
2.2 Amazon Web Services come Cloud Provider 13
2.3 Deep Learning nel Cloud 15
2.3.1 Introduzione al Deep Learning 15
2.3.2 Deep Learning & Cloud Computing 16

3 Tecnologie e Servizi utilizzati 19
3.1 AWS Lamdba 20
3.2 Amazon API Gateway 21
3.3 Amazon DynamoDB 22
3.4 Amazon Simple Storage Service 23
3.5 AWS Identity and Access Management 24
3.6 Amazon Elastic Container Registry 25
3.7 Docker 26
3.7.1 Confronto tra Containers e Virtual Machines 27
3.7.2 Architettura di Docker 28
3.7.3 Dockerfile 29
3.8 NeuralProphet 31
3.8.1 Le serie temporali 31
3.8.2 Introduzione a NeuralProphet 32
3.8.3 Processo di previsione temporale 33

4 Sviluppo del Progetto e Realizzazione del Software 35
4.1 Dump dei dati da SQLite 36
4.2 Inserimento ed estrazione di dati 37

4.3 Creazione dell'immagine Docker 40
4.4 Previsione di serie temporali 42

5 Conclusioni e Sviluppi Futuri 47
5.1 Conclusioni 47
5.2 Sviluppi futuri 47

Bibliografia 49

Elenco delle figure 51

Capitolo 1

Introduzione

Il presente lavoro di tesi tratta dell'implementazione di un servizio cloud al fine di sviluppare un sistema che possa archiviare, analizzare e fornire previsioni riguardo i dati nell'ambito del progetto ReCity.

Il nostro obiettivo primario è quello di progettare e sviluppare un sistema innovativo basato su Serverless Computing[1]. Questo sistema non solo consentirà l'archiviazione e l'estrazione eficiente di dati, ma avrà anche la capacità di addestrare una Rete Neurale, utilizzando tali dati come input, al fine di generare previsioni accurate per dati temporali. Il presente lavoro si occuperà anche di creare un ambiente flessibile e scalabile in modo da gestire in modo eficiente i vari carichi di lavoro.

Il sistema sviluppato offre specifiche funzionalità attraverso l'accesso a API, consentendo agli utenti di interagire direttamente con le risorse e le capacità del sistema. I servizi Cloud utilizzati sono messi a disposizione da Amazon Web Services (AWS), il Cloud più completo e utilizzato al mondo.

[1] Esecuzione di applicazioni senza gestire Server

L'utilizzo di un sistema basato su serverless computing ha rivoluzionato il modo in cui si sviluppano e implementano applicazioni. Questi tipi di sistemi offrono numerosi vantaggi rispetto all'approccio tradizionale basato su server dedicati o macchine virtuali. I possibili vantaggi possono essere riassunti in:

- la scalabilità automatica delle risorse. A differenza dei server dedicati, che richiedono un'infrastruttura fissa, un sistema serverless si adatta dinamicamente alla domanda di risorse;
- la responsabilità di garantire l'afidabilità e la tolleranza ai guasti è delegata al provider di servizi Cloud;
- gli sviluppatori possono concentrarsi esclusivamente sulla logica dell'applicazione, senza preoccuparsi della gestione e della configurazione dell'infrastruttura sottostante;
- permette di ridurre notevolmente i costi operativi. In un'architettura serverless, non è necessario acquistare, configurare e gestire fisicamente i server;

Per la realizzazione di questi tipi di sistemi, si fa afidamento a dei provider Cloud che forniscono una vasta gamma di servizi e soluzioni per soddisfare le esigenze di aziende di qualsiasi dimensione, attraverso una semplice connessione ad Internet. Ogni fornitore ha i propri servizi e vantaggi unici, ma tutti condividono l'obiettivo di fornire risorse informatiche scalabili e afidabili tramite il Cloud.

In questo progetto verranno trattati gli aspetti necessari ad implementare un sistema di questo genere, basato proprio sull'utilizzo di un provider Cloud (AWS) che mette a disposizione risorse e capacità di calcolo per eseguire tutte le funzionalità adottate per lo sviluppo di questo progetto.

1.1 Obiettivi

Il progetto di tesi si pone l'obiettivo di sviluppare un sistema basato su serverless computing che consenta l'eficiente acquisizione e l'inserimento dei dati in un database mediante il processo di data ingestion, in particolare DynamoDB[2], attraverso l'utilizzo di servizi Cloud come API Gateway[3] e AWS Lambda Functions[4]. In seguito, anche la capacità di addestrare una Rete Neurale per la predizione di dati timeseries.

Per questo scopo è stato utilizzata il modello di intelligenza artificiale NeuralProphet[1]; in particolare è stata usata la sua implementazione in Python che fornisce un'implementazione semplice ed eficace di modelli di previsione temporale basati su Reti Neurali. Per lo sviluppo di quest'ultima parte si intende realizzare un immagine Docker che verrà poi messa a disposizione come servizio tramite un endpoint grazie ad un servizio Cloud specializzato nella creazione e distribuzione di API, chiamato API Gateway.

Riassumendo, il flusso di lavoro consiste in:

1. Utilizzare il servizio Amazon DynamoDB per la creazione di un Database NoSQL;
2. Realizzare delle Lambda Functions per andare ad inserire ed estrarre dati, in un dato intervallo temporale, all'interno di DynamoDB relativi al contesto ReCity;
3. Realizzare un immagine Docker che conterrà tutto il codice relativo alla realizzazione e all'addestramento della Rete Neurale.
4. Mettere a disposizione queste funzioni attraverso le API utilizzando il servizio Amazon API Gateway.

Nota: Il linguaggio di programmazione utilizzato è il Python, linguaggio di programmazione ad alto livello multi paradigma.

[2] Database NoSQL fornito da AWS
[3] Servizio che semplifica creazione, gestione ed esposizione di API per le applicazioni
[4] Servizio di calcolo serverless fornito da AWS

1.2 Struttura della Tesi

Il presente lavoro di tesi è strutturato nei seguenti capitoli:

- Apertura

1. Introduzione;
2. Tecnologie Utilizzate;
3. Progettazione e sviluppo del progetto;
4. Conclusione e sviluppi futuri;
5. Appendice.

Capitolo 2

Cloud Computing

Perché sviluppare applicazioni in Cloud?

É sicuramente una domanda che ci si pone durante la lettura di questa tesi.

Il cloud computing permette a chi ne usufruisce di rilasciare ed accedere a servizi e distribuire applicazioni tramite un'infrastruttura messa a disposizione da un'organizzazione che si occupa di mantenerla e gestirla (Provider Cloud). Ciò che rende interessante l'utilizzo di tali tecnologie da parte dei clienti dei Cloud provider e che caratterizza il Cloud come particolare classe di sistema distribuito può essere riassunto in:

- Possibilità di non doversi occupare della gestione dell'infrastruttura fisica, che 'e afidata al Provider Cloud;
- Scalabilità automatica ed altamente dinamica, che permette alle applicazioni Cloud di scalare, sia espandendosi che riducendosi, in funzione del carico di lavoro a cui tali applicazioni sono sottoposte, questo si traduce in un'allocazione di risorse dinamica;
- Modello di pagamento Pay-As-You-Go (PAYG), nasce come metodo di fatturazione che consiste nel fatturare soltanto le risorse effettivamente utilizzate dall'utente, questo permette di non dover pagare risorse che spesso non vengono utilizzate, come succede nel caso di noleggio o acquisto di server, sia virtuali che reali;

- Omogeneità dovuta ad un ambiente di virtualizzazione condiviso che permette di nascondere le differenze, sia a livello hardware che software, che possono esistere tra i vari componenti utilizzati per realizzare concretamente l'architettura del sistema;

Prima dell'avvento del cloud computing, le aziende di qualsiasi dimensione erano costrette a percorrere principalmente due strade: acquistare intere infrastrutture dedicate ai servizi informatici nonché organizzare un luogo fisico dove conservarle in sicurezza (luogo che era spesso una sala server presso la sede dell'azienda stessa), oppure afittare gli spazi fisici offerti dai proprietari di grandi data center per conservare e mantenere attive le proprie infrastrutture.

In entrambi i casi sorgevano grandi ineficienze in termini di utilizzo delle risorse, poiché le aziende dovevano preventivare di quante risorse informatiche avrebbero avuto bisogno ben prima dell'effettiva implementazione di qualunque sistema software.

Volendo cercare di dare una definizione di cosa sia il Cloud, potremmo affermare che esso è un ambiente costituito da un'infrastruttura altamente dinamica tramite la quale le risorse sono distribuite come servizi ('as a Service').

2.1 Modelli di distribuzione dei servizi

Con as-a-Service si indica in genere un servizio di Cloud Computing gestito da un provider esterno per conto dell'utente, che in questo modo ha la possibilità di dedicarsi alle attività strategiche, come la scrittura di codice e le relazioni con i clienti. Ogni tipo di Cloud Computing offre la possibilità di delegare la gestione di un numero sempre maggiore di componenti dell'infrastruttura on premise.

Esistono tre tipologie principali di Cloud Computing as-a-Service, ognuna con un diverso grado di gestione lasciata all'utente:

- Infrastructure as a Service (IaaS)
- Platform as a Service (PaaS)
- Software as a Service (SaaS)

Analizziamo ora le tre tipologie di distribuzione e forniture di servizi informatici da parte dei Provider Cloud.

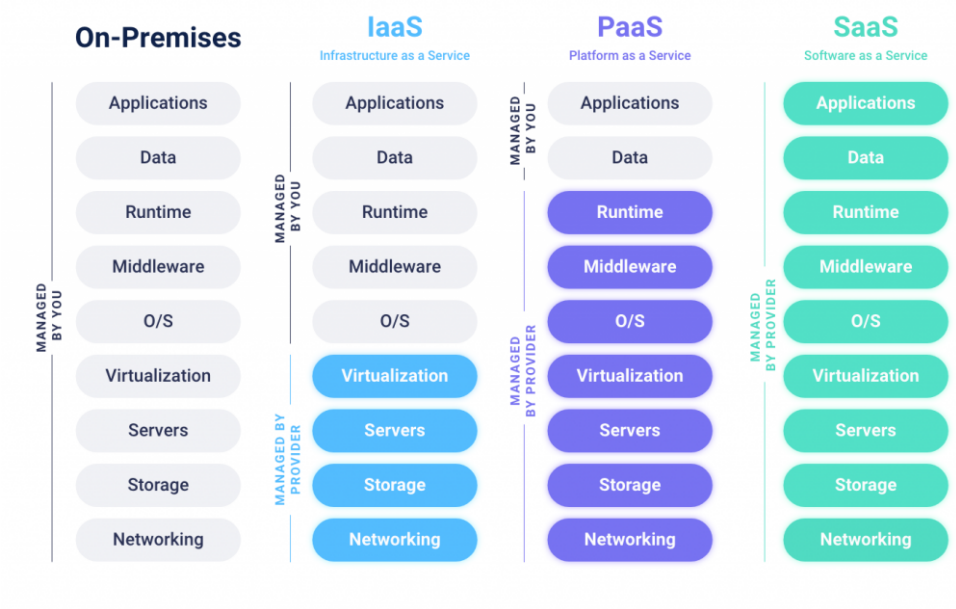

Figura 2.1: Diagramma che rappresenta le varie tipologie di distribuzione di servizi
[2]

2.1.1 Infrastructure as a Service (IaaS)

Il livello più inferiore (IaaS) si occupa di fornire server, memorie di massa ed infrastrutture di rete virtualizzando l'hardware reale.

L'idea che sta alla base di questo tipo di servizio è che i server virtuali di un'infrastruttura elastica dovrebbero essere offerti ad utenti differenti, i quali devono restare isolati gli uni dagli altri, sia per quanto riguarda i loro dati che le performance. Riuscire ad ottenere un buon livello di isolamento può tuttavia risultare complesso, perché in realtà gli utenti condividono l'infrastruttura fisica che si trova sotto lo strato virtualizzato, costituita da componenti di rete, unità di memorizzazione di massa e server.

In questo modo l'utente ha pieno controllo sull'infrastruttura virtuale, potendo configurare e gestire le risorse in base alle proprie esigenze. Possono scalare orizzontalmente (aggiungendo più risorse) o verticalmente (aumentando la capacità delle risorse esistenti) per adattarsi alle esigenze di carico di lavoro. I fornitori di servizi IaaS addebitano gli utenti in base al consumo effettivo delle risorse, offrendo così un modello di pagamento flessibile e riducendo i costi operativi.

2.1.2 Platform as a Service (PaaS)

Il livello intermedio (PaaS) fornisce l'ambiente e gli strumenti più ad alto livel-lo necessari per la realizzazione e l'esecuzione delle applicazioni, appoggiandosi all'infrastruttura del livello sottostante;

L'obbiettivo del PaaS è quello di ospitare i componenti software creati da utenti diversi in un ambiente di esecuzione condiviso, il quale offre loro le funzionalità più comuni ed usate. Si concentra in particolare sulla fornitura di un ambiente di sviluppo e di esecuzione completo per le applicazioni. In questo caso, i fornitori di servizi offrono piattaforme che consentono agli sviluppatori di accelerare il ciclo di sviluppo consentendo di creare, testare e distribuire le proprie applicazioni senza che ci sia la necessità di configurare e gestire l'infrastruttura sottostante.

Per poter far ciò è necessario potenziare ulteriormente le politiche di isolamento, il quale non deve essere pi'u garantito solo a livello di macchine virtuali, ma a livello dei componenti, che possono trovarsi ad essere ospitati ed eseguiti sullo stesso server.

2.1.3 Software as a Service (SaaS)

Il livello più alto (SaaS) è costituito dalle applicazioni vere e proprie e dai servizi di cui esse necessitano.

Il concetto attorno al quale si sviluppa il SaaS è la possibilità di offrire lo stesso Software tramite la rete ad utenti diversi e isolati. I fornitori di servizi metto-no quindi a disposizione applicazioni completamente gestite, che possono essere utilizzate tramite una connessione Internet. Gli utenti accedono alle applicazioni tramite un'interfaccia utente direttamente dal web.

Un fattore molto importante è fare in modo che ogni utente abbia la percezione di essere l'unico utilizzatore del software in questione ma allo stesso tempo l'applicazione stessa deve essere consapevole di avere più utenti diversi che condividono le stesse risorse hardware e software. Questo permette una maggiore eficienza nell'utilizzo delle risorse e riduce i costi per gli utenti finali.

2.2 Amazon Web Services come Cloud Provider

Nel presente capitolo, si esplorerà il motivo per cui AWS (Amazon Web Services) è ampiamente riconosciuto come uno dei principali fornitori di servizi Cloud a livello globale.

AWS è una piattaforma di proprietà del gruppo Amazon che offre servizi di Cloud Computing, elaborazione e distribuzione di contenuti e molto altro, ideali per creare applicazioni sofisticate in modo flessibile, scalabile e afidabile.

Lanciata nel 2002, AWS oggi copre il 58% del fatturato di Amazon facendosi strada come leader nel settore del Cloud Computing, offrendo oggi più di 175 servizi in Cloud completati da data center a livello globale. Milioni di clienti utilizzano AWS per diminuire i costi ed innovarsi in modo più rapido.

Come già anticipato AWS offre una vasta gamma di funzionalità rispetto a qualsiasi altro provider Cloud, dalle infrastrutture per il calcolo, l'archiviazione e i database fino alle nuove tecnologie, quali il Machine Learning, AI [1], Data Lake, Analytics e IoT [2].

I vantaggi principali che AWS offre rispetto ai competitor sono principalmente sulla facilità di utilizzo, costi di utilizzo ed una interfaccia utente semplice ed intuitiva. Di seguito esamineremo alcuni dei vantaggi che si riscontrano utilizzando AWS:

- Facilità d'uso: AWS è progettato per consentire ai fornitori di applicazioni e rivenditori di ospitare le applicazioni in modo rapido e sicuro, che si tratti di un'applicazione esistente o di una nuova applicazione basata su SaaS;
- Flessibilità: AWS permette di selezionare il sistema operativo, il linguaggio di programmazione, la piattaforma di applicazione Web, il database e altri servizi secondo necessità. Inoltre, si avrà a disposizione un ambiente virtuale dove caricare il software e i servizi necessari per l'applicazione da sviluppare;
- Convenienza: i costi si basano esclusivamente sulla potenza di elaborazione, lo storage e le risorse utilizzate.

[1] Artificial Intelligence
[2] Internet of Things

Tra i vari servizi offerti dalla piattaforma se ne vedranno solo alcuni fondamentali utilizzati per la realizzazione del presente lavoro di tesi:

- AWS Lambda;
- Amazon API Gateway;
- Amazon DynamoDB;
- Amazon ECR;
- Amazon S3;
- AWS Identity and Access Management.

Nel prossimo capitolo questi servizi verranno approfonditi adeguatamente e verrà mostrata anche un architettura completa di come questi servizi comunicano tra di loro per lo sviluppo di tale progetto di tesi.

2.3 Deep Learning nel Cloud

2.3.1 Introduzione al Deep Learning

Il Deep Learning è una classe di algoritmi di apprendimento automatico (Machine Learning) che si basa sull'utilizzo di reti neurali artificiali (ANN) per apprendere dai dati, fare previsioni o prendere decisioni. Queste reti neurali sono composte da diverse strati di neuroni artificiali interconnessi, ispirati al funzionamento del cervello umano.

I modelli di Deep Learning sono in grado di riconoscere relazioni complesse tra le variabili, questo le rende particolarmente eficaci in problemi di apprendimento automatico, come riconoscimento di immagini, elaborazione del linguaggio naturale, traduzione automatica, riconoscimento della voce e molto altro per produrre informazioni e previsioni accurate;

L'addestramento di una rete neurale profonda coinvolge l'utilizzo di un ampio dataset di addestramento. Durante questa fase (Forward Propagation), i dati di input vengono alimentati alla rete neurale, che li elabora attraverso i vari strati di neuroni, generando una previsione o una risposta come output. Una volta ottenuto l'output previsto, viene calcolato l'errore tra l'output previsto e il valore corretto desiderato. Nella fase successiva (Backward propagation) l'errore viene propagato dai neuroni dell'output verso i neuroni degli strati nascosti, consentendo di aggiustare i pesi delle connessioni tra i neuroni in modo iterativo in modo da ridurre l'errore complessivo della rete. Ciò permette alla rete di adattarsi ai dati e migliorare progressivamente le sue prestazioni nel compito specifico.

Tuttavia, richiede una grande quantità di dati e risorse computazionali, motivo per cui il Cloud Computing è diventato un'opzione sempre più popolare per eseguire addestramenti di reti neurali profonde su risorse virtualizzate e scalabili.

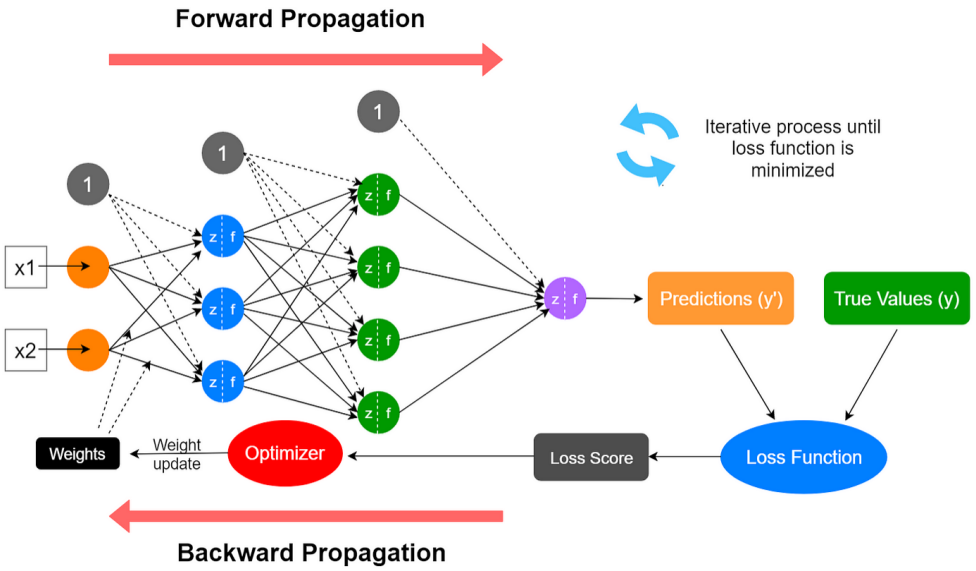

Figura 2.2: Diagramma che rappresenta la fase di addestramento e ottimizzazione della rete
[3]

2.3.2 Deep Learning & Cloud Computing

Come già accennato, il Deep Learning rappresenta una svolta significativa nell'ambito dell'apprendimento automatico, consentendo alle macchine di apprendere e prendere decisioni complesse attraverso le reti neurali artificiali. Tuttavia, l'elaborazione di modelli di Deep Learning richiede risorse computazionali considerevoli, come potenza di calcolo e capacità di archiviazione. In questo contesto, il Cloud Computing emerge come un'infrastruttura ideale per supportare l'implementazione e l'esecuzione eficiente di algoritmi di Deep Learning.

L'utilizzo del Cloud Computing consente di inglobare e gestire in modo semplice grandi dataset per formare algoritmi e consente di scalare i modelli di Deep Learning a costi inferiori utilizzando la potenza di elaborazione della GPU. Sfruttando le reti distribuite, l'apprendimento approfondito sul Cloud consente di progettare, sviluppare e formare le applicazioni di Deep Learning più velocemente.

AWS mette a disposizione molti servizi distinti per la gestione, esecuzione e distribuzione di applicazioni di Deep Learning. Ne vedremo uno in particolare nel prossimo capitolo:

- Elastic Container Registry (ECR)

Approfondiamo ora alcuni dei vantaggi che rendono il Cloud molto conveniente per l'implementazione di complessi algoritmi di Deep Learning ed anche le sfide associate all'utilizzo di questa tecnologia per il Deep Learning.

- Vantaggi:
  - Scalabilità: Le piattaforme cloud offrono accesso a una vasta gamma di istanze di macchine virtuali (VM) con capacità di elaborazione e memorizzazione flessibili. Questo consente agli utenti di adattare le risorse di calcolo alle dimensioni del set di dati e alla complessità del modello, garantendo prestazioni elevate e tempi di addestramento ridotti;
  - Accesso a potenza di calcolo di alto livello: il Deep Learning richiede una grande quantità di potenza di calcolo per addestrare reti neurali profonde complesse. Il Cloud Computing offre l'accesso a risorse di calcolo ad alte prestazioni, come le unità di elaborazione grafica (GPU[3]) e i processori di tipo Tensor, che sono appositamente progettati per le operazioni di Deep Learning. Questi acceleratori hardware consentono un'elaborazione significativamente più veloce rispetto alle risorse di calcolo tradizionali, consentendo ai ricercatori di sperimentare e addestrare modelli complessi in tempi più brevi;
  - Riduzione dei costi: L'utilizzo del Cloud Computing per il Deep Learning può ridurre i costi associati all'acquisto, alla manutenzione e all'aggiornamento dell'hardware dedicato per l'elaborazione di modelli di Deep Learning. Invece di investire in infrastrutture costose, gli utenti possono pagare solo per le risorse di calcolo effettivamente utilizzate e scalare verso l'alto o verso il basso in base alle necessità;

[3]Graphics processing unit

- Sfide:
  - Latenza della rete: L'utilizzo del Cloud Computing per il Deep Learning richiede il trasferimento di grandi quantità di dati tra i server Cloud. La latenza della rete può rappresentare una sfida in quanto influisce sulla velocità di trasferimento dei dati e può rallentare il processo di addestramento del modello. È importante considerare la latenza della rete durante la progettazione del flusso di lavoro del Deep Learning nel Cloud e scegliere una soluzione che riduca al minimo i tempi di trasferimento dei dati.

  - Sicurezza e privacy dei dati: Il Deep Learning spesso coinvolge dati sensibili e proprietari. L'archiviazione e l'elaborazione di tali dati nel Cloud sollevano preoccupazioni sulla sicurezza e la privacy. È fondamentale scegliere un provider di servizi Cloud afidabile che offra misure di sicurezza solide, come la crittografia dei dati, il controllo degli accessi e la conformità alle normative sulla privacy dei dati.

  - Dipendenza dal fornitore di servizi cloud: L'utilizzo del Cloud Computing per il Deep Learning implica una dipendenza dal provider di servizi Cloud scelto. È importante valutare attentamente i contratti di servizio, le politiche di prezzo e le funzionalità offerte dal provider prima di impegnarsi a lungo termine. La migrazione dei dati e delle applicazioni da un provider all'altro può essere complessa, quindi è essenziale pianificare attentamente l'infrastruttura e considerare la portabilità dei dati e dei modelli di Deep Learning.

In definitiva, il Cloud Computing offre un ambiente ideale per il Deep Learning, consentendo agli utenti di sfruttare al massimo il potenziale delle risorse di calcolo scalabili, dei servizi integrati e delle soluzioni flessibili offerte dai provider di servizi Cloud come AWS. Saper navigare le opportunità e le sfide del Cloud per il Deep Learning è essenziale per ottenere risultati ottimali e sfruttare appieno le potenzialità di questa tecnologia in continua evoluzione.

Capitolo 3

Tecnologie e Servizi utilizzati

In questo capitolo si illustrano gli strumenti e i servizi utilizzati per la realizzazione di questo progetto di tesi:

- AWS Lambda: servizio di elaborazione per l'esecuzione di codice.
- Amazon API Gateway: servizio per la creazione di API REST.
- Amazon DynamoDB: servizio di database NoSQL.
- Amazon S3: servizio di archiviazione di oggetti.
- AWS IAM: servizio per la gestione di autorizzazioni.
- Amazon ECR: servizio per la gestione di immagini di container.
- Docker: software per gestire container .
- NeuralProphet: framework per la predizione di dati timeseries.

3.1 AWS Lamdba

AWS Lambda è un servizio di calcolo basato su eventi serverless che permette di eseguire codici per qualsiasi tipo di applicazione o servizio back-end senza effettuare il provisioning o gestire server. Può essere attivato da oltre 200 servizi AWS ed applicazioni di software come servizio (SaaS), pagando solo ciò che viene usato.

Lambda esegue il codice su un'infrastruttura di elaborazione ad alta disponibilità e gestisce tutta l'amministrazione delle risorse di elaborazione, compresa la manutenzione del server e del sistema operativo, il provisioning e la scalabilità automatica della capacità e la registrazione. Con Lambda, tutto quello che occorre fare è fornire il proprio codice in uno dei runtime di linguaggio supportati da Lambda.

Il codice viene organizzato in funzioni Lambda. Il servizio Lambda esegue la funzione solo quando necessario e si dimensiona automaticamente. Verrà addebitato soltanto il tempo di calcolo utilizzato e non verrà addebitato alcun costo quando il codice non è in esecuzione.

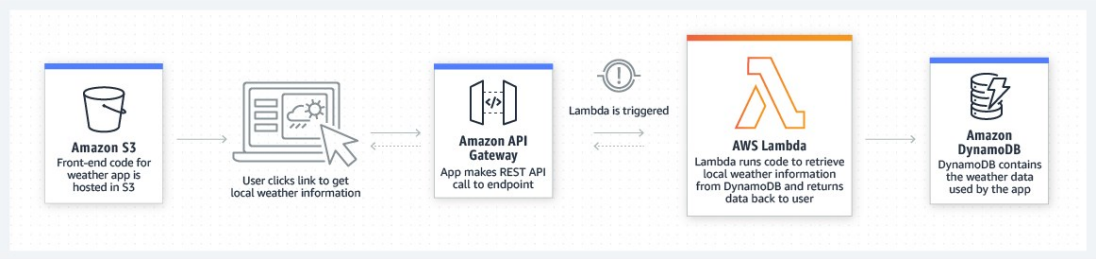

Figura 3.1: Diagramma che rappresenta il flusso di lavoro di una Applicazione Web che invoca AWS Lambda
[4]

3.2 Amazon API Gateway

Amazon API[1] Gateway è un servizio completamente gestito che semplifica per gli sviluppatori la creazione, la pubblicazione, la manutenzione, il monitoraggio e la protezione delle API su qualsiasi scala. Le API fungono da "porta di entra-ta" per consentire l'accesso delle applicazioni ai dati, alla logica aziendale o alle funzionalità dai servizi back-end.

API Gateway consente di creare API RESTful e WebSocket che rendono possibili applicazioni di comunicazione bidirezionale in tempo reale. API Gateway supporta carichi di lavoro containerizzati e senza server, oltre che applicazioni Web.

API Gateway gestisce tutte le attività di accettazione ed elaborazione relative a centinaia di migliaia di chiamate API simultanee, inclusi gestione del trafico, supporto CORS[2], controllo di accessi e autorizzazioni, throttling, monitoraggio e gestione delle versioni delle API. Verrà addebitato solo il costo per le chiamate API ricevute e i volumi di dati trasferiti in uscita, in questo modo è possibile ridurre i costi in funzione dell'utilizzo delle API.

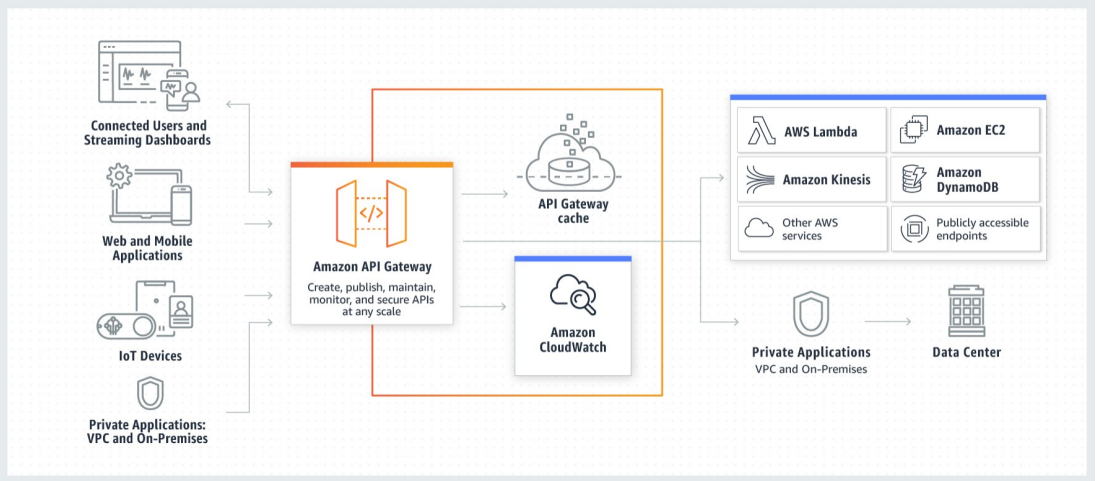

Figura 3.2: Diagramma che rappresenta il flusso di lavoro di per la chiamata di una API
[5]

[1] Application Programming Interface
[2] Cross-Origin Resource Sharing

3.3 Amazon DynamoDB

É un servizio di database NoSQL completamente gestito che combina prestazioni elevate e prevedibili con una scalabilità ottimale e a latenza ridotta adatto per applicazioni che richiedono una rapida memorizzazione e un accesso eficiente ai dati.

DynamoDB utilizza un modello di dati chiave-valore, che significa che i dati sono organizzati in tabelle in cui ogni riga è identificata da una chiave univoca. Supporta anche operazioni di query flessibili tramite indici globali e locali, che consentono di recuperare i dati in base a criteri specifici.

Una delle caratteristiche distintive di DynamoDB è la sua scalabilità automatica. che consente di archiviare e recuperare qualsiasi quantità di dati, per qualunque livello di trafico. Amazon DynamoDB distribuisce automaticamente i dati e il trafico per la tabella su un numero di server suficiente per gestire la capacità di richiesta specificata dal cliente e la quantità di dati archiviati, mantenendo al contempo la coerenza e la rapidità delle prestazioni.

DynamoDB offre anche la possibilità di archiviare i dati in modo duraturo e replicare automaticamente i dati su più zone di disponibilità all'interno di una regione AWS, garantendo una maggiore resilienza e disponibilità.

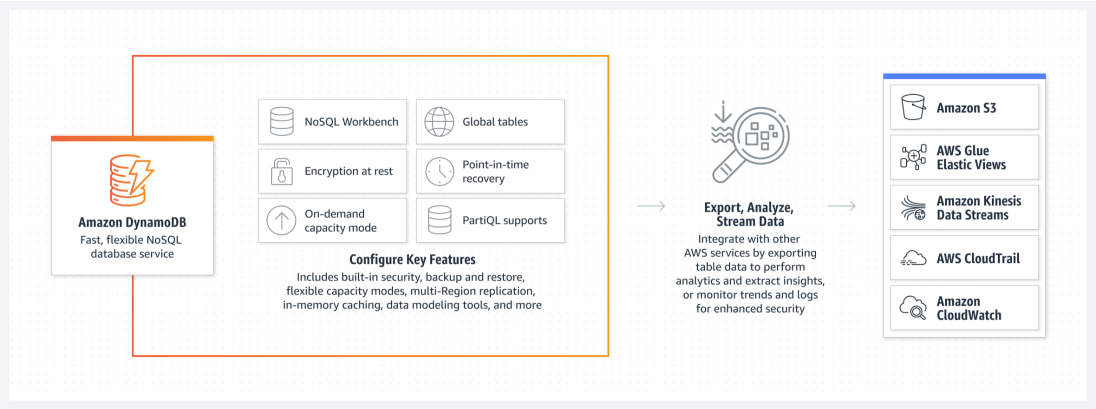

Figura 3.3: Diagramma che rappresenta le caratteristiche principali di DynamoDB e la sua integrazione con altri servizi
[6]

3.4 Amazon Simple Storage Service

Amazon Simple Storage Service (S3) è un servizio di storage Cloud progettato per archiviare e recuperare grandi quantità di dati in modo sicuro, afidabile e scalabile.

S3 offre uno spazio di archiviazione praticamente illimitato e consente di memorizzare oggetti, come file, immagini, video e documenti, all'interno di "secchielli" (buckets) all'interno di una determinata regione AWS. Ogni oggetto è identificato da una chiave univoca e può essere recuperato tramite un URL[3] univoco.

Una delle caratteristiche chiave di S3 è la sua durabilità. I dati memorizzati in S3 sono replicati automaticamente su più dispositivi e strutture all'interno di una regione AWS per garantire l'afidabilità e la disponibilità dei dati.

S3 supporta anche la gestione dei permessi e la crittografia dei dati per garantire la sicurezza delle informazioni archiviate. È possibile definire politiche di accesso granulari, assegnare autorizzazioni agli utenti e crittografare i dati in transito e a riposo utilizzando protocolli di crittografia avanzati.

Figura 3.4: Diagramma che mostra come spostare, gestire i dati archiviati in S3 e come analizzarli utilizzando altri servizi
[7]

[3] Uniform Resource Locator

3.5 AWS Identity and Access Management

AWS Identity and Access Management (IAM) è un servizio di gestione delle identità e degli accessi che consente quindi di gestire in modo sicuro l'accesso ai servizi e alle risorse AWS, consentendo di definire e controllare le autorizzazioni degli utenti e delle applicazioni.

IAM permette di creare e gestire utenti, gruppi, ruoli e politiche all'interno di un account AWS. Ogni utente viene fornito di credenziali di accesso uniche, come un nome utente e una password, o l'uso di certificati di accesso. I gruppi consentono di raggruppare utenti con esigenze di accesso simili, semplificando la gestione delle autorizzazioni. I ruoli consentono di definire privilegi di accesso per le applicazioni o altri servizi AWS.

Le politiche infine definiscono cosa un utente, un gruppo o un ruolo può fare su determinate risorse AWS. Le politiche di accesso sono definite utilizzando un linguaggio di controllo degli accessi basato su JSON [4] (JSON-based Access Control Language), che consente di specificare permessi granulari e restrizioni di accesso.

L'utilizzo di IAM consente di applicare il principio del "modello del privilegio minimo", in cui agli utenti e alle applicazioni vengono forniti solo i privilegi di accesso necessari per svolgere le loro attività, riducendo il rischio di accessi non autorizzati o di abusi.

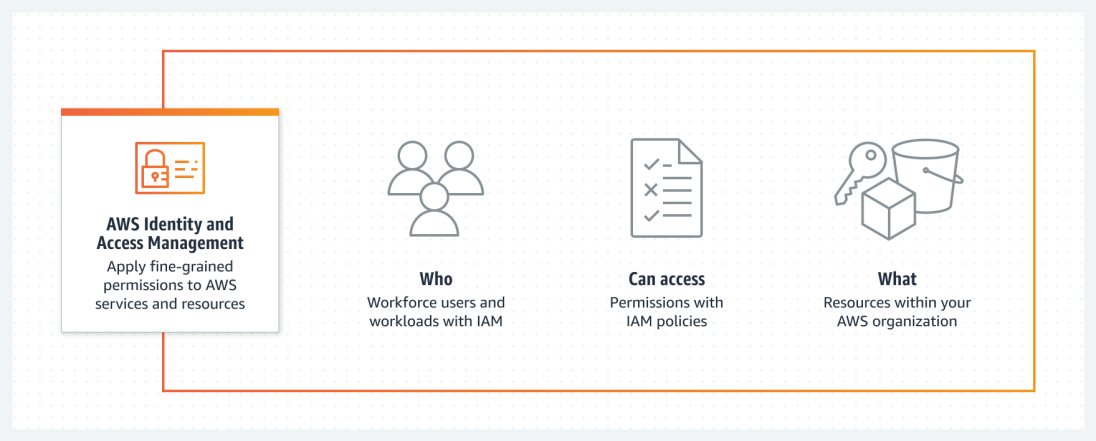

Figura 3.5: Diagramma che mostra il funzionamento di AWS IAM [8]

[4] JavaScript Object Notation

3.6 Amazon Elastic Container Registry

Amazon Elastic Container Registry (ECR) è un servizio di container registry gestito e offerto da AWS. Funziona come un registro privato per l'archiviazione e la gestione delle immagini dei container.

ECR consente di caricare, archiviare e scaricare immagini dei container utilizzate per eseguire applicazioni "containerizzate" su servizi di orchestrazione come ECS[5] o EKS[6] offrendo un'interfaccia semplice ed intuitiva e garantendo allo stesso tempo la sicurezza e la scalabilità.

Amazon ECR offre funzionalità di scalabilità e disponibilità. Può gestire un numero elevato di richieste di pull e push di immagini dei container in modo simultaneo, consentendo un'elaborazione rapida e afidabile. Supporta anche la distribuzione globale delle immagini dei container attraverso la rete di disponibilità regionale di AWS, consentendo un accesso rapido e afidabile alle immagini da parte di servizi e utenti in tutto il mondo.

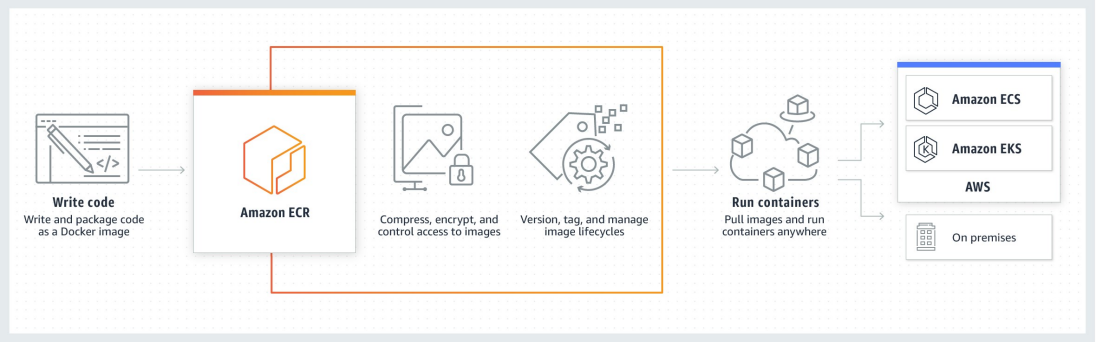

Figura 3.6: Diagramma che mostra il funzionamento di Amazon ECR [9]

[5] Elastic Container Service
[6] Elastic Kubernetes Service

3.7 Docker

Docker è la piattaforma open-source più utilizzata per la creazione di container. In particolare consente di creare, distribuire e gestire applicazioni in modo efficiente.
Essenzialmente, Docker consente di confezionare un'applicazione e tutte le sue dipendenze in un contenitore virtuale chiamato appunto container. Viene utilizzato principalmente per creare software agile, vale a dire consegnare nuove versioni di applicazioni per vari sistemi operativi in modo veloce e sicuro.

Quando viene creato un container, viene creato un ambiente virtuale, basato su Linux, isolato all'interno del sistema operativo host. Questo ambiente virtuale prende il nome di Docker Engine o Docker Daemon e condivide il Kernel del sistema operativo host. I container saranno quindi eseguiti all'interno di questi ambienti virtuali ed è proprio questo che garantisce l'isolamento delle applicazioni.

Un container quindi è un'istanza isolata e portatile di un'applicazione che include tutto il necessario per eseguire correttamente l'applicazione, come codice, runtime, librerie di sistema, variabili d'ambiente e file di configurazione.

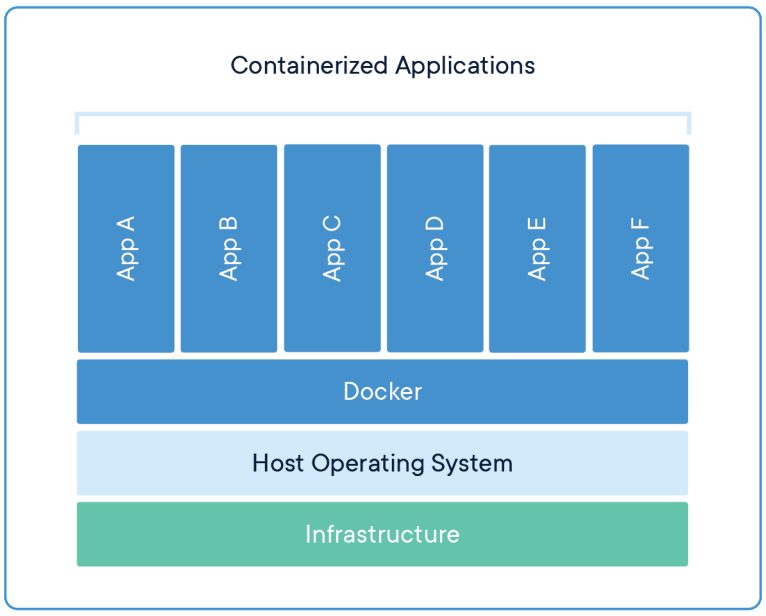

Figura 3.7: Schema di principio di Docker
[10]

3.7.1 Confronto tra Containers e Virtual Machines

Sia i Containers che le Virtual Machines (VMs) hanno pressoché gli stessi benefici per quanto riguarda l'isolamento delle applicazioni, ma lavorano a differenti livelli di virtualizzazione in quanto i Containers virtualizzano il sistema operativo, invece le VMs virtualizzano l'hardware. Questo fa sì che i Containers siano maggiormente portabili ed eficienti.

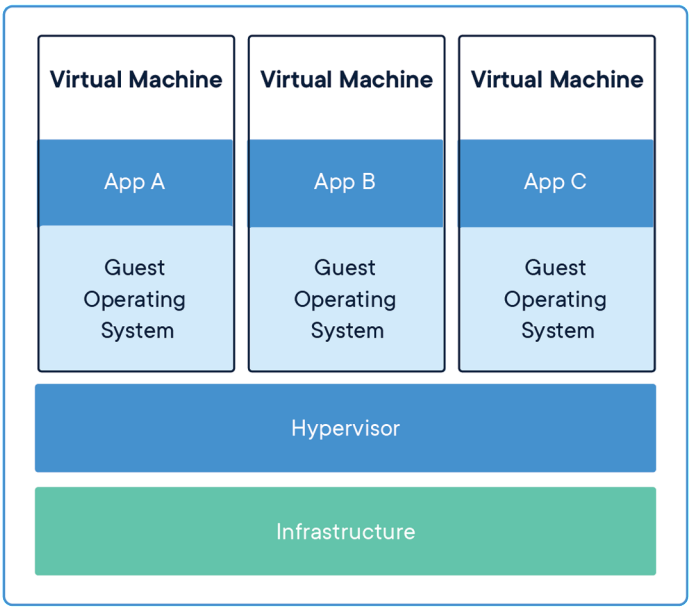

Figura 3.8: Schema di principio delle VMs
[10]

Come risultato abbiamo che possiamo virtualizzare molti sistemi operativi su di uno stesso host. La virtualizzazione dell'intero sistema operativo però richiede un utilizzo di RAM e CPU molto elevato poiché hanno al loro interno moltissime funzionalità che aumentano sempre più.
Nella maggior parte dei casi però queste funzionalità sono inutili per l'applicazione che ci gira su di esso, quindi cresce sempre più la necessità di eliminare dal sistema operativo queste funzionalità afinché l'applicazione desiderata risulti essere più leggera. Per questo scopo risulta essere conveniente l'utilizzo proprio di Containers invece che di VMs.

Un altro importante vantaggio è la dimensione, in quanto per i Containers si parla dell'ordine di qualche decina o centinaia di MegaByte, invece per le VMs si

parla dell'ordine di decine di GigaByte di spazio per cui il loro avvio è decisamente più lento rispetto ad un Containers.

3.7.2 Architettura di Docker

Docker si basa su un'architettura Client-Server composta da tre componenti:

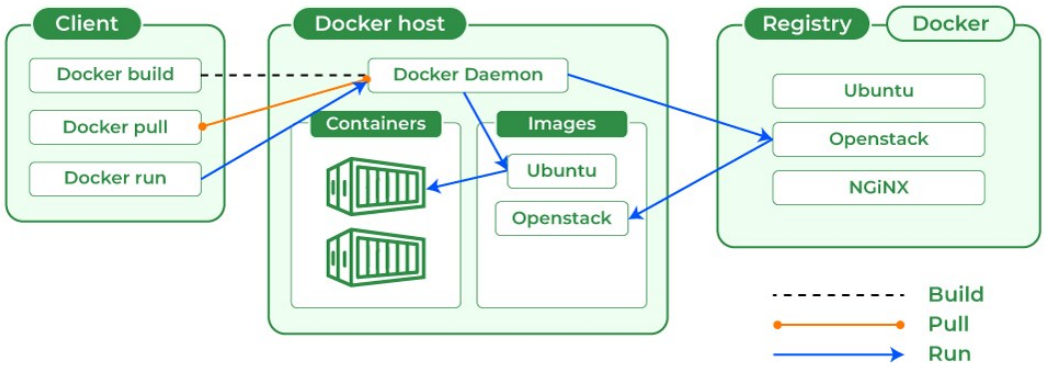

Figura 3.9: Architettura e componenti di Docker
[11]

3.7.2.1 Client Docker

Il Client Docker è l'interfaccia che permette agli utenti di interagire con Docker. Può essere eseguito da linea di comando o attraverso l'utilizzo di una API. Il Client Docker invia comandi al Docker Daemon per creare, eseguire, arrestare e gestire i container. Questo componente ti permette di lavorare con Docker utilizzando comandi intuitivi e semplici da utilizzare. Si può ad esempio creare nuovi container, avviare, arrestare o eliminare container esistenti o creare nuove immagini Docker.

3.7.2.2 Docker host

Il Docker host è l'ambiente di base che ospita e supporta l'esecuzione dei container Docker, ppuò essere un computer fisico, un server virtuale o una macchina virtuale. Il Docker host fornisce l'infrastruttura necessaria per eseguire i container,

inclusi i requisiti di sistema operativo, le risorse hardware, i driver di archiviazione e di rete. Ogni Docker host ha le sue risorse isolate, come CPU, memoria e spazio su disco, che vengono suddivise e assegnate ai container in esecuzione. Quando Docker è installato sul Docker host, il Docker Daemon è avviato come un servizio in background.

Il Docker Daemon (noto anche come Docker Engine) è il componente principale di Docker che gira in background su un host. È responsabile della creazione e gestione dei container. Quando riceve comandi dal Client Docker, il Docker Daemon si occupa di eseguirli. Gestisce il ciclo di vita dei container, compresa la creazione, l'avvio, la sospensione, l'arresto e l'eliminazione dei container. Il Docker Daemon monitora inoltre lo stato dei container, dei volumi e delle reti, garantendo il corretto funzionamento delle applicazioni containerizzate.

3.7.2.3 Docker Registry

Il Docker Registry è un registro centralizzato in cui vengono archiviate le immagi-ni Docker. Un'immagine Docker rappresenta un pacchetto autonomo che contiene tutto il necessario per eseguire un'applicazione, inclusi il codice, le dipendenze, le variabili d'ambiente e i file di configurazione. Il Docker Registry consente agli utenti di caricare (push) e scaricare (pull) immagini Docker dal registro. È possibile utilizzare il Docker Hub, che è il registro pubblico di Docker, oppure è possibile creare registri privati per archiviare e distribuire immagini personalizzate. Il Docker Registry è fondamentale per la condivisione e la distribuzione delle immagini Docker tra diversi ambienti e team di sviluppo.

3.7.3 Dockerfile

Il Dockerfile è un file di testo che contiene le istruzioni per creare un'immagine Docker. È uno dei componenti fondamentali di Docker per definire il processo di build di un'immagine.

Il Dockerfile fornisce una descrizione dettagliata di come l'immagine Docker deve essere costruita, specificando le dipendenze, le configurazioni dell'ambiente e le istruzioni per eseguire determinate azioni durante la fase di build dell'immagine. Queste istruzioni vengono eseguite sequenzialmente dal Docker Daemon per creare un'immagine Docker riproducibile e consistente.

Le principali istruzioni che possono essere utilizzate in un Dockerfile sono:

- FROM: Questa istruzione è obbligatoria e deve essere la prima istruzione presente nel Dockerfile. Specifica l'immagine di base da cui derivare l'immagine. Ad esempio, FROM ubuntu indica che l'immagine sarà basata su Ubuntu.
- RUN: Esegue comandi all'interno dell'immagine durante la fase di build aggiungendo così un nuovo layer all'immagine base. Può essere utilizzata per installare pacchetti, eseguire script o eseguire altre operazioni necessarie per preparare l'immagine.
- COPY: Copia i file e le directory dall'host locale alla posizione desiderata all'interno dell'immagine Docker.
- ENV: Imposta variabili d'ambiente all'interno dell'immagine.
- CMD e ENTRYPOINT: Specificano il comando predefinito da eseguire quando il container viene avviato. La differenza tra i due è che ENTRYPOINT non può essere sovrascritta da nuovi argomenti passati durante l'avvio del container ma vengono aggiunti come argomenti aggiuntivi al comando specificato.
- EXPOSE: Specifica le porte di rete su cui l'applicazione in esecuzione all'interno del container ascolta le connessioni.

3.8 NeuralProphet

3.8.1 Le serie temporali

I dati riguardanti le serie temporali hanno ormai assunto un ruolo predominante nella maggior parte dei settori industriali. Molte aziende infatti hanno migliora-to notevolmente la raccolta dei dati sino al punto che la quantità e disponibilità di essi supera le loro capacità analitiche. Come elaborare e prevedere le serie temporali industriali sta diventando rapidamente un argomento importante, so-prattutto nelle applicazioni in cui le previsioni influenzano decisioni aziendali o operative con conseguenze potenzialmente fatali.

I dati di serie temporali sono un tipo specifico di dati che sono organizzati in sequenze temporali ordinate.Ogni singolo dato in una serie temporale è associato a un timestamp che indica il momento in cui è stato acquisito.

Le serie temporali possono essere unidimensionali, come ad esempio una se-quenza di valori numerici registrati a intervalli di tempo regolari, come la tem-peratura giornaliera, i prezzi delle azioni o il numero di vendite giornaliere di un prodotto. Possono anche essere multidimensionali, con più variabili che ven-gono misurate simultaneamente, ad esempio i dati di sensori che registrano la temperatura, l'umidità e la pressione atmosferica contemporaneamente.

Un aspetto chiave dei dati di serie temporali è che spesso mostrano strutture, comportamenti ricorrenti, tendenze o stagionalità intrinseche che possono esse-re analizzate per ottenere informazioni utili. Alcuni modelli comuni nelle serie temporali includono trend, stagionalità, ciclicità, periodicità e variazioni casuali o rumorose.

L'analisi e la previsione dei dati di serie temporali rivestono un ruolo cru-ciale in molti contesti. Comprendere i modelli e le dinamiche all'interno dei dati può fornire informazioni preziose per prendere decisioni informate, fare previsioni future e identificare anomalie o deviazioni dai comportamenti attesi.

Per analizzare i dati di serie temporali, vengono utilizzate varie tecniche, tra cui modelli di reti neurali come NeuralProphet.

3.8.2 Introduzione a NeuralProphet

NeuralProphet è una libreria open-source basata su Python che è stata sviluppata per la previsione temporale utilizzando modelli di reti neurali. È stata creata come un'estensione del framework di previsione Prophet di Facebook, che è ampiamente utilizzato per la previsione temporale.

NeuralProphet sfrutta il potere delle reti neurali per migliorare la precisione delle previsioni temporali. Utilizza un modello di rete neurale feedforward completamente connesso, noto anche come rete neurale feedforward multistrato o Multilayer Perceptron (MLP). Questo tipo di modello è noto per la sua capacità di apprendere modelli complessi dai dati di input.

Un concetto fondamentale del modello NeuralProphet è la sua struttura modulare. Il modello è composto da moduli, ognuno dei quali contribuisce con un componente additivo alla previsione, come descritto nell'equazione:

$$\hat{y}_t = T(t) + S(t) + E(t) + F(t) + A(t) + L(t)$$

Dove:

- T(t): Tendenza al tempo *t*
- S(t): Effetti stagionali al tempo *t*
- E(t): Effetti degli eventi e delle festività al tempo *t*
- F(t): Effetti di regressione al tempo *t* per variabili esogene conosciute nel futuro
- A(t): Effetti di auto-regressione al tempo *t* basati su osservazioni passate
- L(t): Effetti di regressione al tempo *t* per osservazioni ritardate di variabili esogene

Tutti i moduli dei componenti del modello possono essere configurati singolarmente e combinati per comporre il modello. Se tutti i moduli vengono disattivati, viene adattato solo un parametro di offset statico come componente di tendenza. Di default, sono attivati solo i moduli di tendenza e stagionalità.

NeuralProphet è stato sviluppato con l'obiettivo di fungere da ponte tra i metodi di previsione tradizionali e quelli basati sull'apprendimento automatico. Questo approccio mira a sfruttare i vantaggi di entrambe le metodologie. Basandosi sul modello Prophet, NeuralProphet introduce funzionalità aggiuntive, come la selezione automatica degli iper-parametri durante l'addestramento e la possibilità di incorporare un componente auto-regressivo gestito dal modello di rete neurale AR-Net[7][12].

Il modello AR-Net è una rete neurale completamente connessa, in cui l'output istantaneo $\hat{y}_t$ corrisponde al valore al tempo t, mentre gli input sono costituiti dai p valori precedenti della serie temporale y_{t-1}, ..., y_{t-p}. In caso di previsioni a più passi (previsioni per più punti temporali futuri), è possibile aggiungere ulteriori strati nascosti e/o nodi di output.

Un aspetto importante di NeuralProphet è la sua capacità di modellare il contesto locale utilizzando la regressione auto-regressiva e la regressione co-variata. Questo approccio svolge un ruolo fondamentale nel trattamento delle serie temporali in cui il futuro a breve termine dipende dallo stato attuale del sistema.

3.8.3 Processo di previsione temporale

NeuralProphet utilizza una rete neurale per generare previsioni basate sulle informazioni estratte dalla decomposizione dei trend. La rete neurale è progettata per apprendere i pattern complessi presenti nei dati e per adattarsi ai cambiamenti nel tempo. Durante il processo di addestramento, NeuralProphet ottimizza i pesi delle connessioni tra i nodi della rete neurale al fine di ridurre al minimo l'errore tra le previsioni e i valori reali.
Vediamo ora in dettaglio il processo di generazione delle previsioni con NeuralProphet, analizzando le fasi di pre-processing dei dati, decomposizione dei trend, generazione delle previsioni, ottimizzazione dei parametri, valutazione delle previsioni e iterazione per il miglioramento continuo.

1. Pre-processing dei dati: Prima di generare previsioni, NeuralProphet richiede un pre-processing dei dati. Ciò include l'elaborazione dei dati temporali, l'imputazione dei valori mancanti e la normalizzazione dei dati, se necessario. NeuralProphet può gestire dati con intervalli di tempo irregolari, ma preferisce dati equidistanti per una migliore performance.

[7] AutoRegressive Neural Network

2. Decomposizione dei trend: NeuralProphet utilizza una variante della decomposizione di Fourier per identificare i diversi componenti del trend nei dati. I componenti di trend possono includere la tendenza generale, le stagionalità annuali, le stagionalità settimanali e le vacanze. La decomposizione dei trend consente a NeuralProphet di separare i diversi fattori che contribuiscono alle fluttuazioni dei dati.

3. Generazione delle previsioni: Una volta decomposti i trend, NeuralProphet utilizza una rete neurale per generare previsioni precise. La rete neurale è composta da più strati, con nodi interconnessi che trasformano gli input in output predetti. Durante l'addestramento della rete neurale, NeuralProphet ottimizza i pesi delle connessioni tra i nodi per minimizzare l'errore tra le previsioni e i valori reali.

4. Ottimizzazione dei parametri: NeuralProphet utilizza un algoritmo di ottimizzazione per trovare i parametri del modello che massimizzano l'accuratezza delle previsioni. Questi parametri includono l'architettura della rete neurale, i pesi delle connessioni, gli iper-parametri dell'algoritmo di ottimizzazione e altre configurazioni specifiche del modello.

5. Backtesting: NeuralProphet consente anche di eseguire il back-testing delle previsioni, valutando le prestazioni del modello sulle dati di addestramento precedenti a una determinata data. Questo aiuta a valutare la bontà del modello e a identificare eventuali problemi di adattamento e sovradattamento (caso in cui il modello si adatta eccessivamente ai dati di addestramento, includendo anche il rumore o le fluttuazioni casuali presenti nei dati).

6. Valutazione delle previsioni: Una volta addestrato il modello, NeuralProphet può essere utilizzato per generare previsioni future. Queste previsioni possono essere valutate utilizzando misure di valutazione delle prestazioni come l'errore medio assoluto (MAE), l'errore quadratico medio (RMSE) o il coeficiente di determinazione (R^2).

7. Fine-tuning e iterazione: NeuralProphet consente anche di iterare e migliorare il modello tramite il fine-tuning. È possibile aggiustare i parametri del modello, modificare l'architettura della rete neurale o apportare altre modifiche per ottenere previsioni più accurate.

Infine, è importante sottolineare che NeuralProphet è un modello estremamente flessibile e può essere personalizzato in base alle specifiche esigenze del problema di previsione temporale. Si ha inoltre la possibilità di configurare diversi aspetti del modello, come l'architettura della rete neurale, i parametri di ottimizzazione e le modalità di decomposizione dei trend.

Capitolo 4

Sviluppo del Progetto e Realizzazione del Software

L'obiettivo di questo progetto è quello di implementare un servizio Cloud basato su REST API per consentire l'accesso a risorse e servizi come l'upload e l'ottenimento dei dati e la possibilità di addestrare una Rete Neurale in grado di predire dati di serie temporali.

É possibile suddividere il lavoro in quattro parti: la prima parte in cui si scaricano (dump) i dati in locale da un db[1] chiamato SQLite; la seconda parte in cui si implementano le funzioni lambda per consentire l'upload e l'ottenimento di dati che vengono messi a disposizione come servizi da API; la terza parte in cui si addestra una Rete Neurale che successivamente verrà containerizzata utilizzando Docker; infine la quarta parte in cui si esegue l'upload dell'immagine Docker su Cloud che verrà messa a disposizione da due API: una per lo start dell'esecuzione che riguarda l'addestramento della Rete Neurale e una per ottenere i dati predetti una volta che verrà completata l'esecuzione.

I dati che prendiamo in considerazione riguardano sensori per la misurazione dell'impedenza elettrica, che sono stati impiantati in vari campioni di malta. Inizialmente le misurazioni venivano salvate attraverso un servizio di back-end basato su Python, e i dati memorizzati in un database SQLite locale, accessibile da remoto. I dati che principalmente ci interessano sono l'ID del campione, la data e l'ora della misurazione e il valore della misurazione.

[1] Database

4.1 Dump dei dati da SQLite

Innanzitutto SQLite è un sistema di gestione di database relazionali, in particolare, è una libreria software che fornisce un database SQL completo, senza il bisogno di un server client-server separato.

Come prima cosa si è dovuto implementare una funzione che effettuasse il dump in locale da questo db secondo una certa ora, data e campione. Si è scelto il linguaggio Python, nonché il linguaggio di scripting più popolare e utilizzato al mondo.

La funzione Python in questione quindi, una volta effettuato il dump in locale, prepara i dati nel formato corretto4.2 per essere salvati poi in DynamoDB attraverso una API. I dati vengono prima filtrati secondo la data e l'ora, poi vengono strutturati in formato JSON ed infine viene aggiunto un nuovo attributo, ovvero Timestamp, ottenuto dalla conversione della data e dell'ora in Epoch Unix[2].

Una volta fatto ciò, i dati vengono formattati nel modo corretto imposto da DynamoDB e inviati attraverso una richiesta HTTPs utilizzando una API per la loro memorizzazione; nel prossimo paragrafo vedremo come è stata creata.

Voci restituite (300)

Operazioni | Crea voce

ID	Timestamp	Com_Port	Date	Frequency	Magnitude	Magnitude_devstd	Magnitude_devstd_outlier
T1_TU03L	1667490301	COM11	03-11-2022	1000	24121.91	0.87	5.21
T1_TU03L	1667490308	COM11	03-11-2022	2000	23881.43	22.26	22.35
T1_TU03L	1667490315	COM11	03-11-2022	3000	23768.81	23.4	23.26
T1_TU03L	1667490323	COM11	03-11-2022	4000	23685.25	0.28	9.53
T1_TU03L	1667490330	COM11	03-11-2022	5000	23657.59	12.86	15.06
T1_TU03L	1667490337	COM11	03-11-2022	6000	23647.45	21.95	21.83
T1_TU03L	1667490344	COM11	03-11-2022	7000	23673.84	0	2.57
T1_TU03L	1667490351	COM11	03-11-2022	8000	23682.31	0.66	7.99
T1_TU03L	1667490359	COM11	03-11-2022	9000	23740.51	0.66	5.05
T1_TU03L	1667490366	COM11	03-11-2022	10000	23820.43	0.75	3.59
T1_TU03L	1667490373	COM11	03-11-2022	11000	23870.32	0.78	10.91
T1_TU03L	1667490380	COM11	03-11-2022	12000	23992.2	8.68	12.21
T1_TU03L	1667490387	COM11	03-11-2022	13000	24066.5	0.94	7.15
T1_TU03L	1667490395	COM11	03-11-2022	14000	24194.09	9.76	9.76
T1_TU03L	1667490402	COM11	03-11-2022	15000	24337.05	7.22	9.63
T1_TU03L	1667490409	COM11	03-11-2022	16000	24435.71	4.46	10.91

Figura 4.1: Dati salvati su DynamoDB

[2]Sistema per rappresentare il tempo con un numero intero

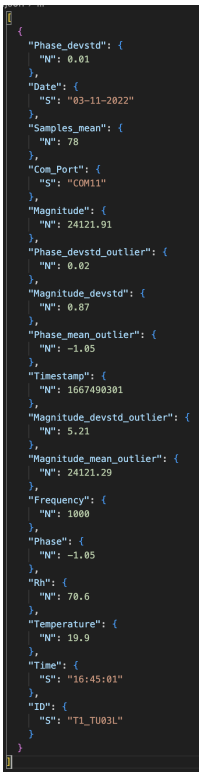

Figura 4.2: Formato dei dati necessario per l'upload su DynamoDB

4.2 Inserimento ed estrazione di dati

Per consentire l'utilizzo delle funzionalità del nostro sistema, sono state realizzate delle API, attraverso il servizio Amazon API Gateway. La prima API che è stata implementata, come menzionato in precedenza, è stata quella per consentire di effettuare l'upload su DynamoDB dei dati scaricati in locale.

Per la realizzazione di questa API è stato necessario realizzare prima una Lambda Function che consentisse l'archiviazione su DynamoDB e poi integrare l'API con essa.

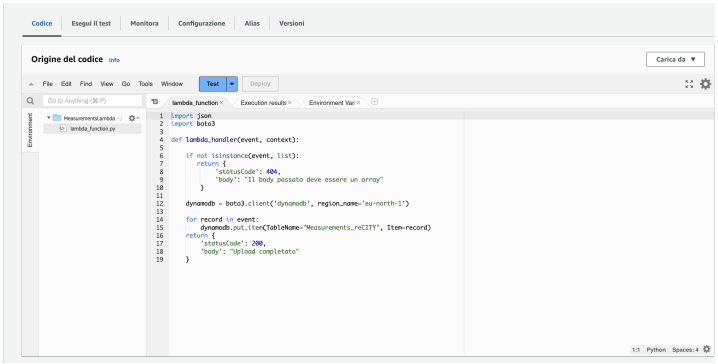

Figura 4.3: Lambda Function per l'archiviazione di dati in DynamoDB

A questo punto si è voluta distribuire l'API per il test della funzionalità. In questa API non sono presenti parametri nell'URL della richiesta, deve essere però specificato il corpo della richiesta dove verranno inseriti tutti i dati da archiviare su DynamoDB all'interno di un vettore, in un determinato formato, per questo motivo la richiesta è di tipo POST[3]. All'esecuzione della richiesta, i dati, vengono passati come parametro alla funzione. I possibili risultati sono due che indicano se la richiesta è stata eseguita correttamente o non correttamente. Per il test delle funzionalità è stato utilizzato Postman[13], piattaforma API per l'utilizzo e la creazione di API.

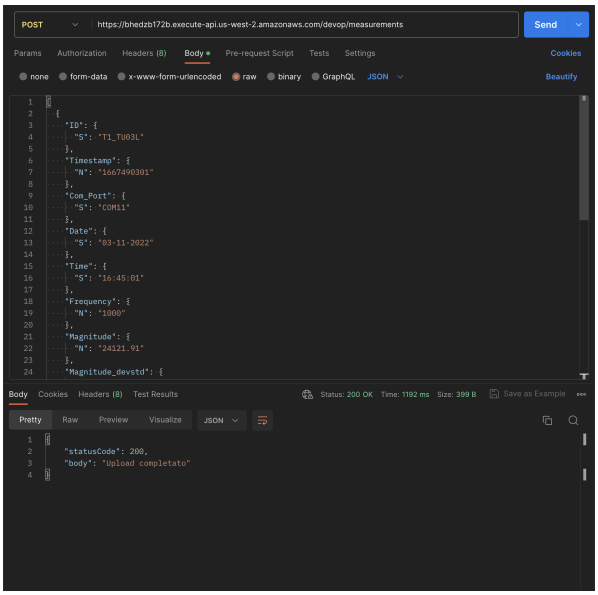

Figura 4.4: Test dell'API per l'upload di dati su DynamoDB eseguito su Postman

[3]Metodo per l'invio dei dati non direttamente nell'URL ma nel corpo nella richiesta

A questo punto si è voluta implementare un'altra Lambda Function che questa volta però consentisse di estrarre i dati dal db. Questi dati devono essere filtrati secondo alcune specifiche: l'ID del campione, la data di inizio e di fine di cui vogliamo vedere i dati ed il numero di campioni che vogliamo mostrare. Oltre il numero massimo di campioni scelto, verrà mostrata la stessa chiamata API ma con un parametro aggiunto automaticamente che consente di visualizzare gli altri campioni successivi.

L'API utilizzata è la stessa del caso precedente ma non verrà utilizzato un metodo POST bensì un metodo GET[4]. In questo caso infatti non verrà aggiunto nessun corpo della richiesta ma i parametri descritti in precedenza. Questi parametri vengono poi passati alla funzione che preleva dal db soltanto i dati che soddisfano quella richiesta.

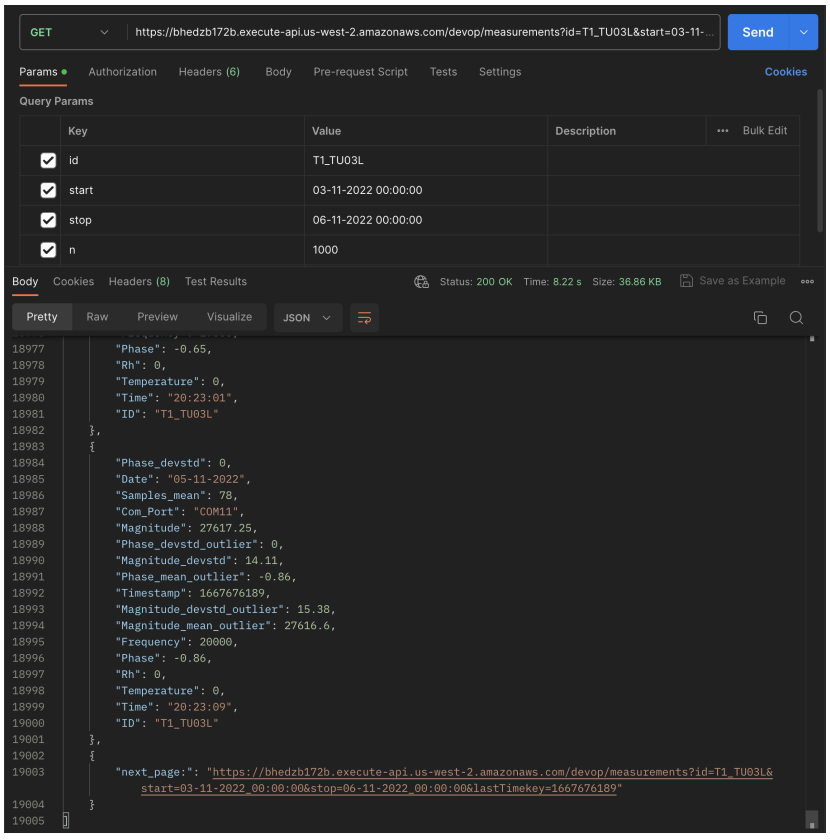

Figura 4.5: Test dell'API per l'estrazione di dati da DynamoDB eseguito su Postman

[4]Metodo per l'invio dei dati direttamente come parametri della richiesta

4.3 Creazione dell'immagine Docker

Per la realizzazione dell'immagine Docker, prima di tutto si è dovuto creare un Dockerfile che consentisse l'integrazione con AWS, infatti AWS e Docker, hanno collaborato per realizzare un'esperienza per sviluppatori semplificata che permette di implementare e gestire direttamente i container su Amazon ECR utilizzando gli strumenti Docker. Nel Dockerfile, in particolare, è stato necessario specificare: la directory della nostra funzione contenente la Rete Neurale, l'immagine di ba-se da cui partire (nel nostro caso Python3.8), l'installazione delle dipendenze di compilazione per aws-lambda-cpp (libreria per sviluppare funzioni AWS Lambda in C++), l'installazione del pacchetto awslambdaric, (componente di runtime per le funzioni AWS Lambda scritte in Python), ed infine, l'installazione dei pacchetti utilizzati, contenuti all'interno del file di testo "requirements.txt".[14]

```
# Define function directory
ARG FUNCTION_DIR="/app"

FROM python:3.8 as build-image

# Install aws-lambda-cpp build dependencies
RUN apt-get update && \
  apt-get install -y \
  g++ \
  make \
  cmake \
  unzip \
  libcurl4-openssl-dev

# Include global arg in this stage of the build
ARG FUNCTION_DIR
# Create function directory
RUN mkdir -p ${FUNCTION_DIR}

# Copy function code
COPY app/* ${FUNCTION_DIR}

# Install the runtime interface client
RUN pip install \
        --target ${FUNCTION_DIR} \
        awslambdaric

# Multi-stage build: grab a fresh copy of the base image
FROM python:3.8

# Include global arg in this stage of the build
ARG FUNCTION_DIR
# Set working directory to function root directory
WORKDIR ${FUNCTION_DIR}

# Copy in the build image dependencies
COPY --from=build-image ${FUNCTION_DIR} ${FUNCTION_DIR}
COPY requirements.txt ${FUNCTION_DIR}
COPY tmp .

RUN pip3 install -r requirements.txt --target ${FUNCTION_DIR}

ENTRYPOINT [ "/usr/local/bin/python", "-m", "awslambdaric" ]
CMD [ "app.handler" ]
```

Figura 4.6: Dockerfile

Abbiamo incluso non solo il Dockerfile come menzionato precedentemente, ma anche un file chiamato requirements.txt contenente l'elenco dei pacchetti utilizzati nel nostro progetto, insieme alle rispettive versioni. Questi pacchetti includono:

- NeuralProphet: una libreria che fornisce un'implementazione di modelli di previsione basati su reti neurali per la previsione di serie temporali.
- Pandas: una rinomata libreria open-source in Python ampiamente utilizzata per la manipolazione e l'analisi dei dati.
- Boto3: una libreria di programmazione fornita da AWS che ci consente di interagire con i servizi di AWS, come ad esempio DynamoDB, utilizzando Python.

Con l'inclusione di questi pacchetti nel nostro progetto, siamo in grado di sfruttarne le funzionalità e le capacità per soddisfare i requisiti specifici legati alla manipolazione, all'analisi e all'interazione con i dati di serie temporali e i servizi di AWS come DynamoDB.

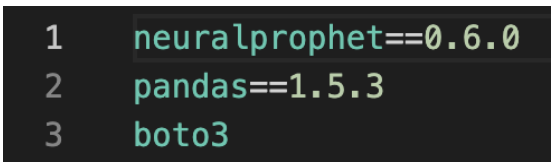

Figura 4.7: File Requirements.txt contenete i pacchetti utilizzati

Una volta realizzato il Dockerfile, si è passati all'implementazione della Rete Neurale (vedremo nel dettaglio come è stata implementata nel prossimo paragrafo) e poi alla creazione dell'immagine Docker. Questa immagine è stata poi caricata su Amazon ECR attraverso l'interfaccia a riga di comando di AWS, chiamata AWS CLI[5].

[5]Command Line Interface

4.4 Previsione di serie temporali

Per la realizzazione del servizio che consente di fornire previsioni di serie temporali su certi dati passati in input, è stato necessario realizzare due API, due Lambda Function ed utilizzare il servizio Amazon S3 ed il servizio AWS Step Functions. Il motivo per cui sono stati utilizzati S3 e Step Function derivano dai limiti del servizio API Gateway, in particolare, ogni richiesta API non può superare il limite di 29 secondi e poiché l'addestramento richiedeva un tempo maggiore, si è pensato di utilizzare Step Function.

Come prima cosa, si è implementata una Lambda Function che consentisse di: salvare i dati passati come input all'interno di Amazon S3, invocare Step Function per l'addestramento della Rete Neurale ed infine restituire un URL dove verranno mostrati le previsioni di serie temporali una volta che Step Function avrà completato l'operazione.

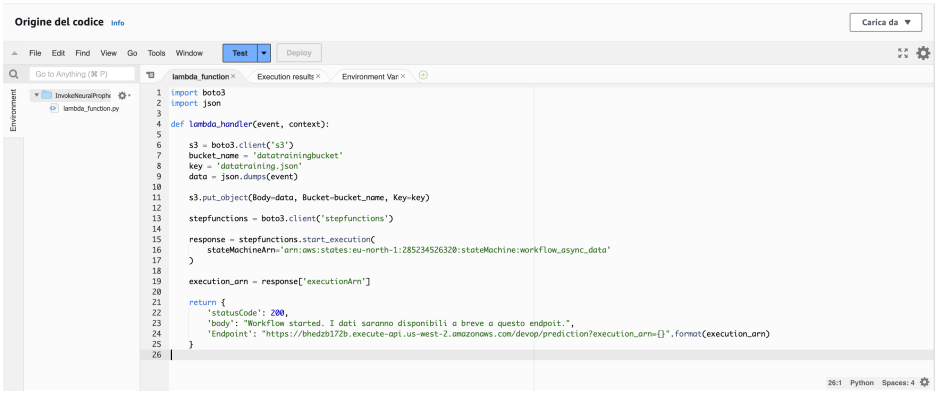

Figura 4.8: Lambda Function che invoca l'esecuzione di Step Function

L'API che si occupa di eseguire questa funzione lambda è di tipo POST, in questo caso si avrà come parametro, il numero di intrevalli temporali da predire e come corpo della richiesta la serie temporale con cui la rete neurale verrà addestrata.

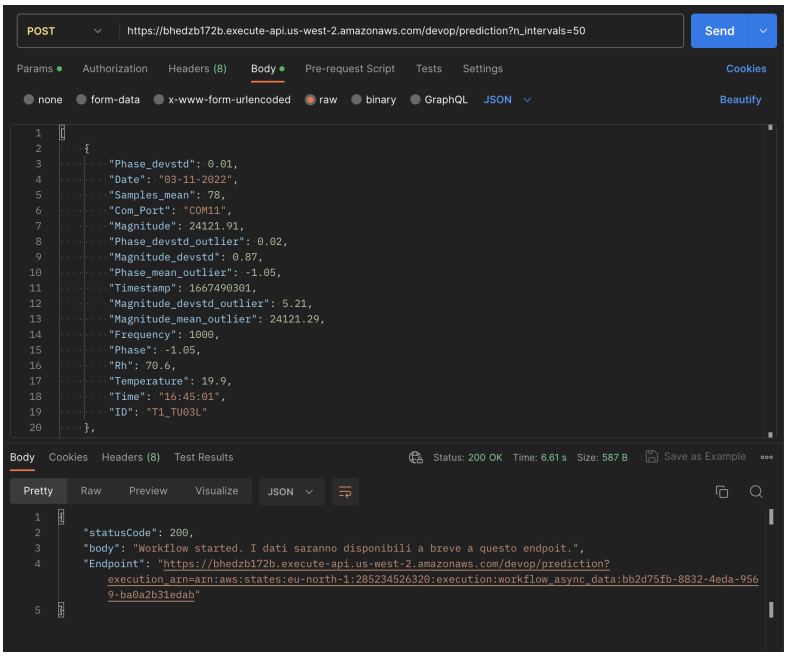

Figura 4.9: Test dell'API per l'invocazione di Step Function per l'addestramento della Rete Neurale eseguito su Postman

Successivamente si è descritto, attraverso il formato JSON, cosa Step Function dovesse eseguire una volta invocata. Nel nostro caso la funzione da eseguire è proprio la Rete Neurale contenuta nell'immagine Docker caricata su Amazon ECR.

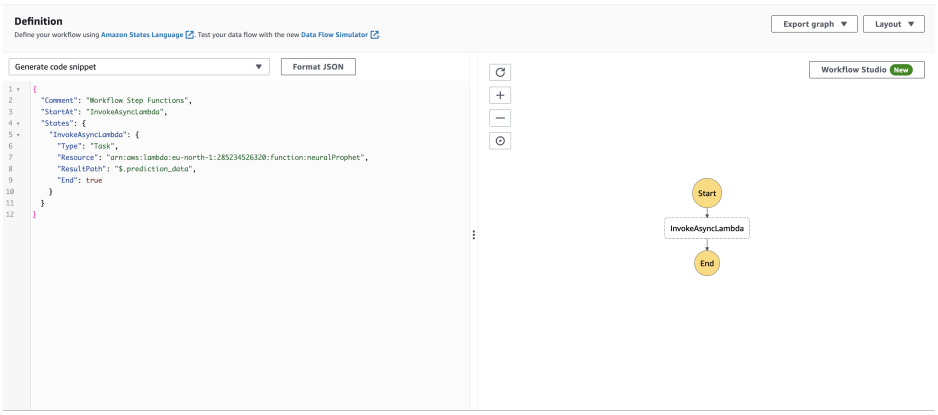

Figura 4.10: Definizione del flusso di lavoro di Step Functon

L'immagine Docker in questione, contiene una funzione che non ha alcun input ma accede ai dati direttamente grazie a S3. Una volta recuperati i dati, questi vengono formattati correttamente per essere dati in input a NeuralProphet per il suo addestramento. Una volta che la Rete Neurale è addestrata, viene restituita le serie temporale predetta.

Infine, per la visualizzazione della previsione effettuata dalla Rete Neurale, si è realizzata un'altra Lambda Function che consente di vedere in tempo reale lo stato dell'esecuzione di Step Function. Questa Lambda Function viene eseguita dall'URL descritto in precedenza.

Per la visualizzazione dei dati ci vorrà circa qualche secondo che varia sia in base alla scelta del numero di intervalli da predire e sia dalla lunghezza della serie temporale passata alla funzione per addestrare la rete neurale. Si consiglia di utilizzare un set di dati per l'addestramento abbastanza grande per una predizione ottimale.

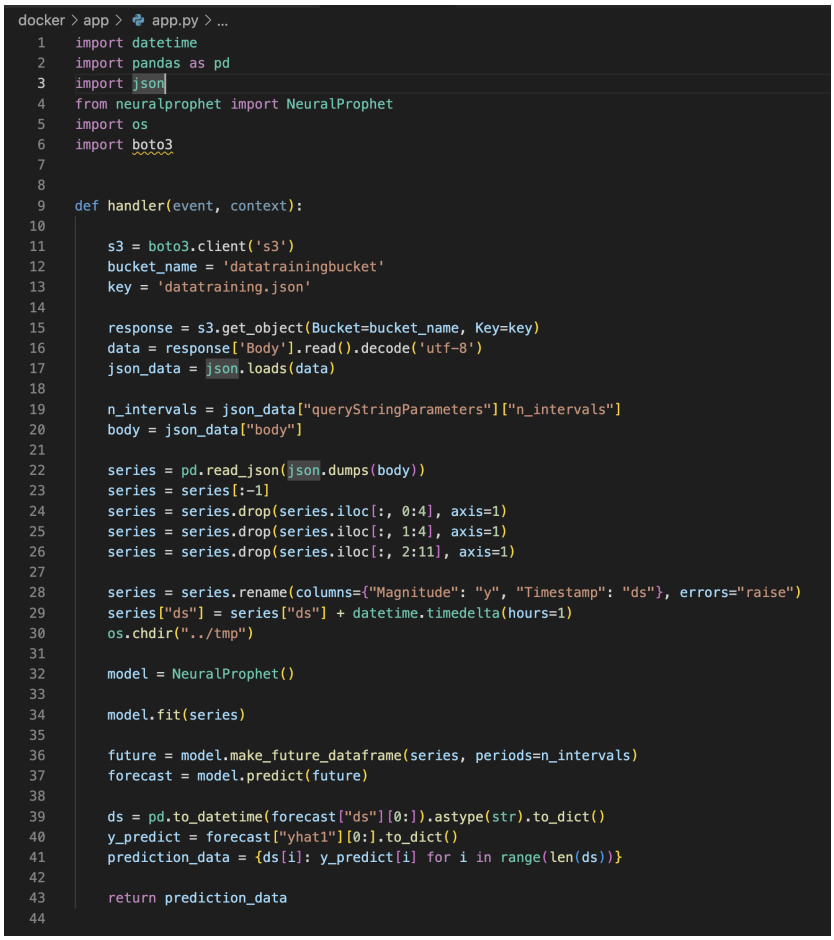

Figura 4.11: Implementazione della Rete Neurale per la previsione di serie temporali

L'API che si occupa di eseguire questa funzione lambda è di tipo GET e si ha come parametro un identificatore univoco specifico per un'istanza di esecuzione di un workflow definito da StepFunctions e questo viene passato automaticamente dall'API precedente per far riferimento alla corrente esecuzione di StepFunctions.

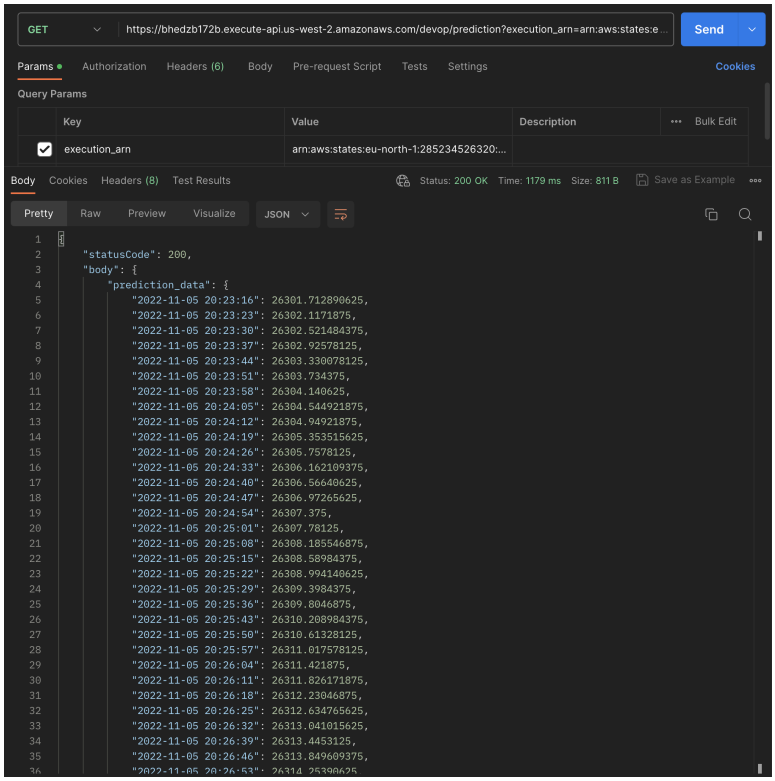

Figura 4.12: Previsioni di serie temporali restituiti dalla Rete Neurale eseguito su Postman

Capitolo 5

Conclusioni e Sviluppi Futuri

5.1 Conclusioni

Il sistema sviluppato, ciò nonostante, ha dato le basi per la creazione di un sistema basato su cloud per agevolare la gestione e l'analisi di dati temporali complessi. Utilizzando una combinazione di servizi è stato creato un ambiente scalabile e flessibile per la messa a disposizione di API per fare data ingestion e l'addestramento di una rete neurale per le previsioni di serie temporali. Il linguaggio Python si è rilevato particolarmente adatto per questo scopo grazie alla sua flessibilità e versatilità.

Sostengo di aver trovato l'argomento di tesi molto interessante affrontando argomenti nuovi e stimolanti i quali hanno portato ad interessarmi particolarmente allo sviluppo di sistemi Cloud. Sono estremamente soddisfatto di quanto appreso e realizzato.

5.2 Sviluppi futuri

Diversi sono i possibili sviluppi futuri di questo progetto di tesi:

- Ottimizzazione degli iperparametri del modello NeuralProphet per migliorare ulteriormente le previsioni e l'accuratezza dei risultati.

- Implementazione di un front-end per avere un'interfaccia semplice ed intuitiva per l'addestramento del modello e l'utilizzo del servizio.
- Ampliare le funzionalità del sistema aggiungendo nuove API o servizi per supportare ulteriori casi d'uso.

Bibliografia

[1] https://neuralprophet.com/contents.html.

[2] https://www.artifakt.com/blog/paas/
paas-vs-iaas-vs-saas-differences-pros-and-cons/.

[3] https://medium.com/data-science-365/
overview-of-a-neural-networks-learning-process-61690a502fa.

[4] https://aws.amazon.com/it/lambda/.

[5] https://aws.amazon.com/it/api-gateway/.

[6] https://aws.amazon.com/it/dynamodb/.

[7] https://aws.amazon.com/it/s3/.

[8] https://aws.amazon.com/it/iam/.

[9] https://aws.amazon.com/it/ecr/.

[10] https://www.docker.com/resources/what-container/.

[11] https://www.geeksforgeeks.org/architecture-of-docker/.

[12] https://ai.facebook.com/blog/
ar-net-a-simple-autoregressive-neural-network-for-time-series/.

[13] https://www.postman.com/product/tools/.

[14] https://docs.aws.amazon.com/lambda/latest/dg/python-image.html.

Elenco delle figure

2.1 Diagramma che rappresenta le varie tipologie di distribuzione di servizi 10

2.2 Diagramma che rappresenta la fase di addestramento e ottimizzazione della rete 16

3.1 Diagramma che rappresenta il flusso di lavoro di una Applicazione Web che invoca AWS Lambda 20

3.2 Diagramma che rappresenta il flusso di lavoro di per la chiamata di una API 21

3.3 Diagramma che rappresenta le caratteristiche principali di Dyna-moDB e la sua integrazione con altri servizi 22

3.4 Diagramma che mostra come spostare, gestire i dati archiviati in S3 e come analizzarli utilizzando altri servizi 23

3.5 Diagramma che mostra il funzionamento di AWS IAM 24

3.6 Diagramma che mostra il funzionamento di Amazon ECR 25

3.7 Schema di principio di Docker 26

3.8 Schema di principio delle VMs 27

3.9 Architettura e componenti di Docker 28

4.1 Dati salvati su DynamoDB 36

4.2 Formato dei dati necessario per l'upload su DynamoDB 37

4.3 Lambda Function per l'archiviazione di dati in DynamoDB 38

4.4 Test dell'API per l'upload di dati su DynamoDB eseguito su Postman 38

4.5 Test dell'API per l'estrazione di dati da DynamoDB eseguito su Postman 39

4.6 Dockerfile 40

4.7 File Requirements.txt contenete i pacchetti utilizzati 41

4.8 Lambda Function che invoca l'esecuzione di Step Function 42

4.9 Test dell'API per l'invocazione di Step Function per l'addestra-mento della Rete Neurale eseguito su Postman 43

4.10 Definizione del flusso di lavoro di Step Functon 43

4.11 Implementazione della Rete Neurale per la previsione di serie temporali 45

4.12 Previsioni di serie temporali restituiti dalla Rete Neurale eseguito su Postman 46