FaceReq, il chatbot serverless

A.Boccini, a.boccini10@gmail.com

Abstract—*I servizi AWS di Amazon offrono molteplici possibilità nell'utilizzo delle risorse. Alta disponibilità, tolleranza ai guasti e manutenibilità sono solo alcuni dei vantaggi offerti da questa solida infrastruttura. Proprio facendo leva su tali elementi si è ideato un sistema capace di orchestrare diversi servizi con lo scopo di offrire un assistente in grado di dialogare con l'utente e di interagire con i servizi di Amazon.*

1 Introduzione

Nell'attuale panorama della programmazione i servizi AWS giocano un ruolo fondamentale. A partire dal non lontano 2006 Amazon offre servizi di ogni tipo agli sviluppatori, alle aziende di piccola, media e grande taglia, agli studenti e ai ricercatori accademici. Ma qual è il motivo di questo successo? Come primo elemento è necessario menzionare il meccanismo con cui i servizi sono offerti: l'utente paga con meccanismi a grana estremamente fine, limitando lo spreco di risorse economiche. Inoltre, le fasi di setup sono notevolmente facilitate o, addirittura, automatizzate. In ultimo non è possibile non menzionare la facilità con cui i numerosissimi servizi comunicano tra loro tramite meccanismi di accesso sempre più raffinati e adattati ai contesti di utilizzo.

Alla luce di questo strumento molto potente è nata la volontà di costruire un chatbot, cioè un assistente testuale che permetta di interagire con diversi servizi AWS, dagli spazi di storage come DynamoDB e S3 fino a servizi più complessi come Rekognition e Lex. Ma il cuore di tale progetto è l'utilizzo di funzioni serverless tramite il servizio AWS Lambda.

Nel seguito verranno descritti i vantaggi e i concetti fondamentali del serverless, che rappresenta il cuore del nostro progetto, per poi spiegare il funzionamento dei servizi AWS utilizzati e mostrare la loro interazione reciproca. Infine verrà brevemente presentata l'interfaccia offerta agli utenti e alcune brevi considerazioni finali.

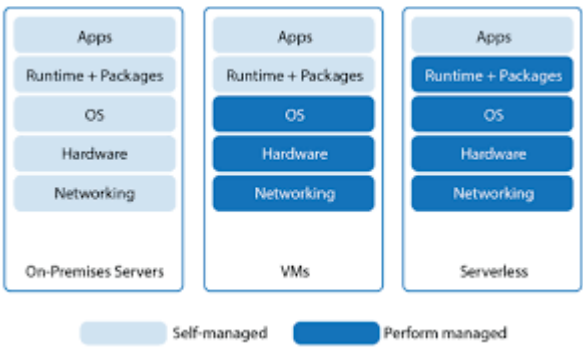
2 Serverless

2.1 Come funziona

Come viene spiegato in [1], il serverless è "un modello di cloud computing in cui gli sviluppatori di applicazioni non devono occuparsi del provisioning dei server o della scalabilità delle proprie app". Quindi ci troviamo di fronte ad una forte facilitazione dello sviluppo di applicazioni, in cui il programmatore può prettamente concentrarsi sulla logica vera e propria senza occuparsi di aspetti "secondari" (anche se, in pratica, sono tutt'altro che secondari). Come scritto in [1], il serverless "consente agli sviluppatori di dedicarsi, non tanto all'infrastruttura, ma al codice". I server che eseguono le computazioni sono gestite dal cloud provider in modo completamente trasparente allo sviluppatore. Inoltre, fatto di primissimo rilievo, la fatturazione riguarda i soli istanti in cui la computazione richiesta occupa il tempo di processamento del server cloud.

Stiamo parlando di un servizio che offre trasparenza allo sviluppatore, risparmio economico e tempi di deployment minimi, oltre a consentire al programmatore di aumentare la propria produttività.

Alla luce di queste considerazioni, ci possiamo chiedere: perchè non usare unicamente il serverless? La risposta non è banale e, soprattutto, potrebbe essere non compresa da chi non sfrutta a pieno le potenzialità della programmazione client-server. Infatti in molti contesti è utile (se non necessario) avere il controllo della logica del server: potrebbe essere necessario dover adattare dinamicamente l'infrastruttura al contesto dell'applicazione oppure dover superare le limitazioni che un cloud provider potrebbe imporre come policy. Altro fattore, non meno importante, è quello legato alla forte dipendenza che si crea con lo specifico cloud provider che offre i server: il passaggio ad altri provider potrebbe generare problemi molto complessi, se non irrisolvibili, in assenza di grandi cambiamenti nella logica dell'applicazione.



Alla luce di queste considerazioni è possibile affermare che la computazione serverless è di gran lunga la soluzione migliore in casi in cui non è necessario avere un controllo sull'infrastruttura server e in cui l'applicazione non richiede particolari condizioni per l'esecuzione. Al contrario, nei casi in cui lo sviluppatore deve controllare a grana fine il contesto della propria applicazione, sono ancora più efficienti metodi "tipici", come l'utilizzo di macchine virtuali, di container o di veri e propri cloud privati.

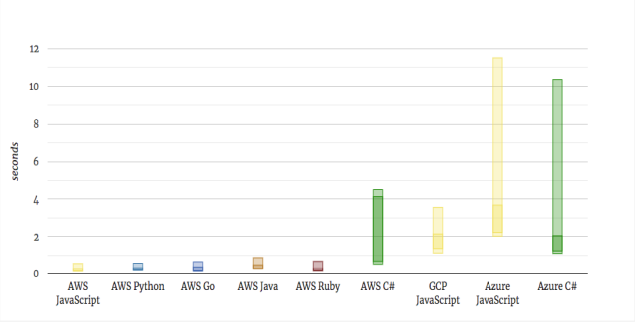
2.2 Analisi prestazioni

Passando ai risultati numerici, confrontiamo la computazione serverless con quella effettuata tramite container.

Prima di tutto, seguendo la spiegazione di [2], il mondo dei

container permette l'utilizzo di una più vasta gamma di linguaggi di programmazione. Inoltre permette di implementare servizi persistenti mentre nel mondo serverless ciò che è eseguito è solo la singola funzione richiesta al server in un determinato istante. Perciò, sempre seguendo [2], è più utile considerare queste due tecnologie come complementari e non come alternative.

Iniziando ad entrare più dettagliatamente nel contesto della nostra applicazione, possiamo notare che i diversi linguaggi di programmazione (ben pochi rispetto al panorama completo) si adattano in modo profondamente diverso al serverless computing.



Typical cold start durations per language

Anche alla luce di questa caratteristica, si è scelto di sviluppare l'applicazione interamente in linguaggio Python (3.8).

3 Servizi AWS

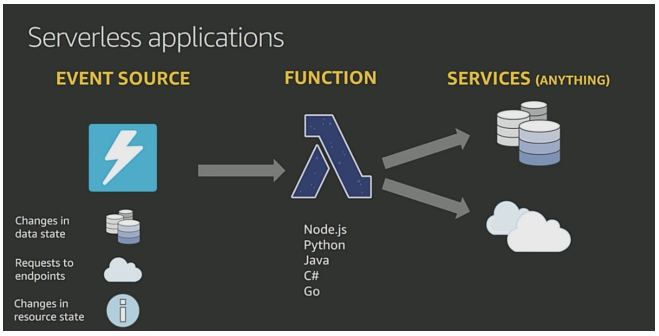
3.1 Lambda

Come descritto in [3], AWS Lambda è il servizio serverless di Amazon. Permette di eseguire task di ogni tipo, dal caricamento di pagine web al processamento di flussi di dati. Facendo riferimento ai risultati prestazionali spiegati in precedenza, AWS sfrutta i container per eseguire le funzioni Lambda dell'utente. Ciò avviene in modo dinamico: prima dell'esecuzione della funzione, l'infrastruttura alloca RAM e CPU necessarie al corretto funzionamento. Al termine dell'esecuzione AWS moltiplica il tempo totale di processamento (in millisecondi) per la tariffa imposta e restituisce all'utente il costo finale. Un elemento estremamente interessante di AWS Lambda è la possibilità di eseguire parallelamente una stessa funzione serverless.

Uno svantaggio di AWS Lambda è quello che riguarda il "cold start time". Infatti, nei casi in cui la funzione non venga eseguita per più di 10-15 minuti, si genera un ritardo tra l'arrivo dell'evento "trigger" e l'esecuzione della funzione, causato dal tempo di inizializzazione del container.

Nello sviluppo dell'applicazione un limite che inizialmente ha generato problemi è quello inerente le dimensioni massime della funzione da eseguire: con un limite di soli 50 MB lo sviluppo con linguaggi di programmazione più onerosi (ad esempio Java) è molto complicato a causa della natura stessa dei linguaggi e della presenza di dipendenze esterne.

Concettualmente, AWS Lambda permette di caricare funzioni "handler", che cioè gestiscono eventi di input. Tali eventi sono detti "trigger" e possono consistere in inserimenti in tabella di DynamoDB, eliminazioni di oggetti in S3 o in invocazioni dirette da parte dell'utente.



Input e output di tali funzioni, nel nostro caso, sono oggetti JSON contenenti le informazioni richieste.

3.2 S3

S3 è il servizio Amazon di memorizzazione di dati. La sua struttura è organizzata in "bucket", che fungono da directory, e in "object", cioè i file veri e propri. Nel contesto della nostra applicazione S3 è il contenitore delle immagini che verranno successivamente analizzate e l'interazione con tale servizio è svolta tramite API apposite messe a disposizione da Amazon.
Uno dei punti di forza di S3 è l'illimitatezza dello spazio di storage, che consente di non porre limiti allo sviluppatore.

3.3 DynamoDB

DynamoDB è un servizio di storage di Amazon che fornisce database NoSQL agli sviluppatori. Basandosi sui principi BASE, viene assicurata un'elevata scalabilità. Il servizio offre la possibilità di generare tabelle, al cui interno vengono ospitati "items". Tali oggetti devono avere una "primary key" e, possibilmente, una "sort key". Nell'ambito della nostra applicazione una tabella DynamoDB consente di memorizzare gli output del servizio Rekognition (che analizzeremo tra breve) e funge da sorgente di informazioni per il chatbot che interagisce con l'utente finale.

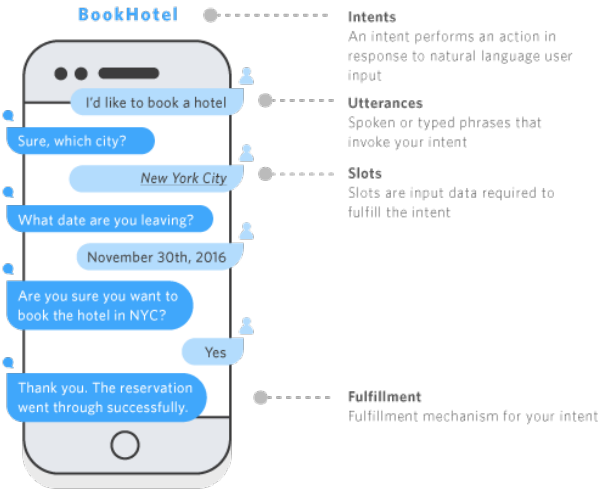
3.4 Rekognition

Il servizio Rekognition mette a disposizioni una vasta gamma di librerie utili all'analisi dettagliata di immagini. Utilizza principi di machine learning che consentono di individuare espressioni facciali, oggetti presenti, possibili proprietà di soggetti ed oggetti. L'architettura è scalabile e consente di avere risultati in brevissimo tempo.

Per la nostra applicazione tale servizio è essenziale: infatti, a partire dalle immagini caricate in fase iniziale, FaceReq inserisce le informazioni inerenti il soggetto dell'immagine all'interno di una tabella DynamoDB (emozioni, età stimata, sesso ecc.)

3.5 Lex

Amazon Lex è un servizio che permette di implementare la medesima interfaccia utilizzata in Alexa all'interno di applicazioni esterne. Offre la possibilità di creare un assistente vocale e testuale interagente con i servizi AWS.



Sfrutta algoritmi di deep learning per analizzare opportunamente gli input dell'utente e per scegliere la risposta più adatta al contesto.
Il servizio si basa su un insieme di "intent", cioè di ambiti di conversazione (affitto appartamento, valutazione di film, stima dell'età ecc.) in cui poter classificare, con una certa probabilità, gli input dell'utente. All'interno di ogni "intent" è possibile specificare degli "utterance", cioè delle frasi tipiche inerenti il contesto specificato (per l'affitto di appartamenti si può scegliere "quanto costa l'appartamento 21?" oppure "vorrei affittare l'appartamento per una settimana"). In ogni "intent" sono specificati degli "slot", cioè delle variabili inerenti l'"intent" specifico, che devono essere inizializzate per permettere al chatbot di generare una risposta valida.

4 FaceRek

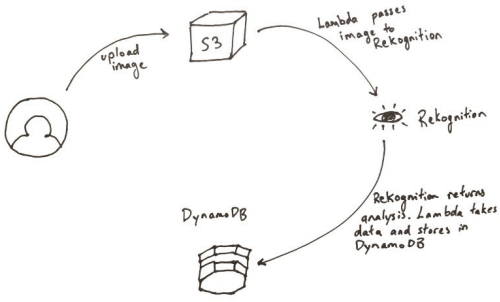
Dopo aver presentato i vari mattoni della nostra applicazione, è giunto il momento di vedere il quadro d'insieme. Per rispettare la modularità del codice verranno affrontati i diversi stadi del processo che porta all'interazione del chatbot con l'utente. Tutte le funzioni menzionate sono state sviluppate in Python 3.8 utilizzando le API offerte da Amazon. L'idea di base dell'applicazione è stata offerta da [4], nonostante essa si basi su un linguaggio di programmazione differente e sull'utilizzo del servizio Alexa Skill Kit per implementare il lato utente. L'interfaccia grafica, in HTML, è stata prelevata da uno dei numerosi esempi offerti da Amazon per poi essere modificata in base alle esigenze specifiche.

4.1 Ambiente di sviluppo

L'applicazione è stata sviluppata in ambiente Unix (Ubuntu 20.04) tramite l'editor di testo Nano (non è stato utilizzato alcun IDE in modo da mantenere le funzioni serverless limitate nella dimensione). La macchine possiede una RAM di 4 GB, una CPU Intel Celeron a 2 core da 2.13 Ghz.

4.2 Preparazione del sistema

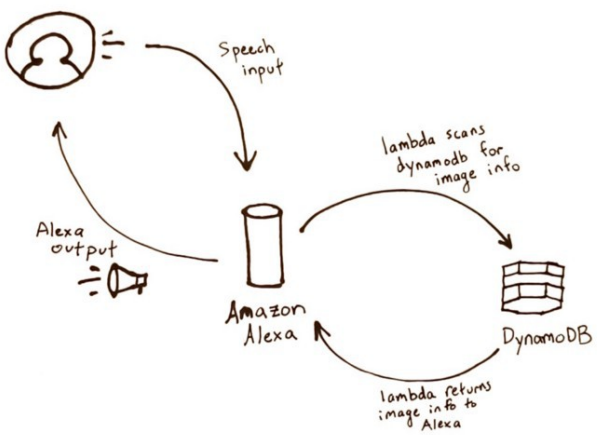
Uno script specifico consente di popolare lo storage S3 tramite API AWS apposite. L'inserimento standard prevede il caricamento delle immagini (in formato jpg) presenti nella cartella "images". Inoltre è possibile specificare, come primo parametro dello script, una specifica immagine da caricare. Tale processo consente di caricare unicamente file jpg e, in caso di file con medesimo nome, verrà effettuata la sostituzione del file presente in S3 con il nuovo file specificato.



Una funzione Lambda specifica (categorize), attivata in seguito ad ogni azione di inserimento in S3, consente di inviare l'immagine caricata al servizio Rekognition, il quale effettuerà l'analisi e restituirà l'output. A questo punto, la stessa funzione Lambda memorizzerà tali output (età stimata, minima e massima, sesso stimato, emozione del soggetto stimata, tutti con la relativa confidenza della stima) in una tabella di DynamoDB.

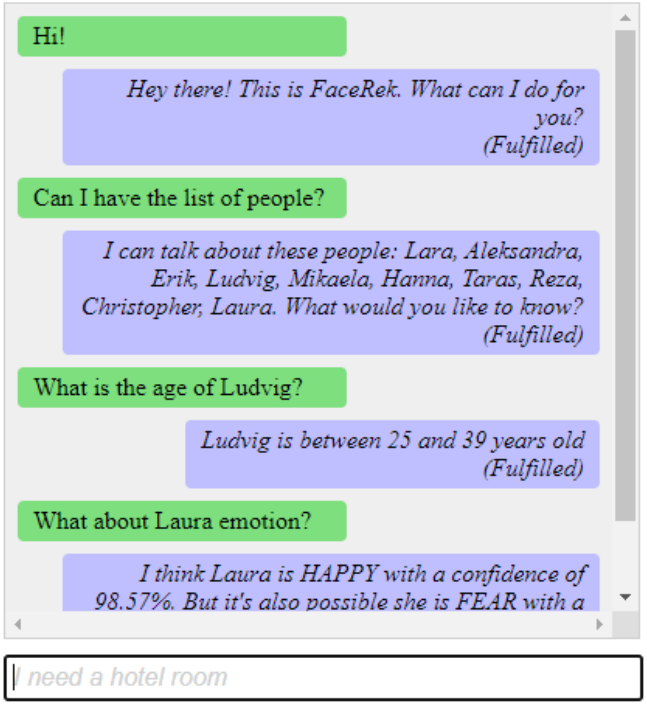
4.3 Interazione utente

Una volta costruito il back-end dell'applicazione, tramite il servizio Amazon Lex, è stata costruita la logica dell'assistente che interagisce con l'utente finale. Sono stati generati degli "intent" specifici riguardanti l'età, le emozioni e il sesso dei soggetti. Due funzioni Lambda, "vision" e "imageResponse" gestiscono gli ambiti di conversazione tra chatbot e utente,



generando la risposta più appropriata. Entrambe le funzioni accedono alla tabella di DynamoDB e estraggono le informazioni necessarie all'utente.

Un'interfaccia HTML consente all'utente di interagire via browser a FaceRek.



In figura è possibile osservare un esempio di interazione tra utente e FaceRek. In particolare, come viene descritto all'utente in una breve descrizione, l'interazione avviene specificando l'ambito della richiesta con il nome del soggetto di interesse.

5 Considerazioni finali

Lo sviluppo di questa applicazione si è rivelata profondamente istruttiva. Ha messo in luce alcune delle numerose potenzialità offerte dall'intero framework di Amazon, adatto all'attuale panorama di sviluppo. Una delle problematiche affrontate durante lo sviluppo riguarda la gestione dei privilegi: nei casi in cui l'interazione avviene esclusivamente tra servizi AWS il problema è gestito in modo opportuno e trasparente da Amazon, ma nel caso di interazione dell'utente finale con il chatbot è stato necessario l'utilizzo del servizio Cognito di Amazon. Esso consente di creare dei ruoli con specifici privilegi, in modo da proteggere le risorse non accessibili e permettere all'utente di godere a pieno dei servizi offerti.
Una considerazione che durante lo sviluppo ha più colpito riguarda le potenziali enormi difficoltà che sarebbero sorte in assenza dei servizi utilizzati: tale esperienza ha sottolineato come Amazon permetta al programmatore di concentrarsi quasi esclusivamente sulla logica dell'applicazione, non preoccupandosi di scalabilità, tolleranza ai guasti, gestione degli accessi ecc.

Riferimenti

[1] https://www.redhat.com/it/topics/cloud-native-apps/what-is-serverless/
[2] https://www.sumologic.com/blog/serverless-vs-containers/
[3] https://www.serverless.com/aws-lambda
[4] https://medium.com/@Kalefive/making-alexa-even-smarter-with-other-aws-services-d8bee368e3f8