Progetto B2: FAAS Management

Corso di Sistemi Distribuiti e Cloud Computing

Matteo Coni
Università Degli Studi di Roma Tor Vergata
Facoltà di Ingegneria Informatica
Matricola 0333880
matteo.coni99@gmail.com

***Abstract*—Questo documento ha l'obiettivo di descrivere la realizzazione di un sistema di gestione FAAS e le scelte progettuali. Verranno presentate l'architettura del sistema e l'implementazione, evidenziando i motivi delle varie scelte e le sfide incontrate durante la realizzazione di questo progetto.**

***Index Terms*—container, docker, aws, lambda, offloading**

I. Introduzione

FAAS Management è un'applicazione sviluppata per permettere agli utenti di usufruire di alcune funzioni che in determinati casi possono essere computazionalmente onerose, senza preoccuparsi dell'architettura sottostante.

Fig. 1. Homepage

È stata sviluppata principalmente in Python e prevede:

- una GUI per scegliere la funzione da eseguire, inserire un input e visualizzare il risultato
- un server che riceve la richiesta dal client e la processa

Quando il server riceve la richiesta (attraverso una chiamata http), sceglie se eseguire la computazione in locale oppure effettuare un offloading su AWS Lambda in base a delle politiche scelte. Ogni funzione verrà eseguita all'interno di un diverso container, tutti instaziati sul server quando richiesto. Attraverso un file di configurazione iniziale è possibile, oltre a scegliere le porte da utilizzare, inserire le credenziali per poter utilizzare AWS Lambda.

II. Tecnologie e librerie adottate

A. Tecnologie

Per lo sviluppo del progetto si è fatto uso delle seguenti tecnologie:

- **Python**: linguaggio di programmazione utilizzato per lo sviluppo del front-end e della logica applicativa
- **Docker**: per il supporto alla virtualizzazione e la gestione dei container
- **Http**: per la comunicazione client-server e server-container
- **AWS Lambda**: come servizio di calcolo serverless offerto da Amazon Web Services

B. Librerie

È stato invece necessario importare e utilizzare le seguenti librerie Python:

- **Flask**: framework web utilizzato per la creazione del server e per la gestione delle richieste-risposte http
- **Tkinter**: libreria Python utilizzata per lo sviluppo del front-end
- **Boto3**: per utilizzare AWS Lambda e permettere l'offloading delle funzioni
- **Json**: libreria per lavorare con il formato JSON (JavaScript Object Notation). Offre metodi per serializzare oggetti Python in formato JSON oppure deserializzare il formato JSON in oggetti Python
- **Requests**: libreria HTTP che semplifica l'invio di richieste HTTP ad altri server e l'elaborazione delle risposte, permettendo metodi come PUT, GET, POST ed altri.
- **Numpy**: libreria Python per il calcolo scientifico, utile in particolar modo per array multidimensionali e matrici.
- **Pytz**: libreria Python per la timezone

III. Descrizione dell'architettura

L'architettura dell'applicazione, come si può vedere in figura 2, prevede un client, un server, e il servizio AWS Lambda. All'interno del client troviamo la GUI, ovvero l'interfaccia attraverso la quale l'utente si interfaccia con il server per le esecuzioni delle funzioni. Per il progetto sono state scelte le seguenti funzioni:

- **Torre di Hanoi**: è un noto problema matematico che coinvolge tre aste (pioli) e un numero variabile di dischi, di dimensione diversa dato, in input. L'obiettivo è

spostare tutti i dischi dal piolo di partenza a quello di destinazione, uno alla volta, e senza poter posizionare un disco più grande su uno più piccolo. La funzione restituisce tutti i passi necessari per risolvere il problema nel minor numero di mosse possibili .

- **Merge Sort**: data in input una sequenza di numeri interi, la funzione restituisce la sequenza ordinata utilizzando l'algoritmo "Merge Sort". Quest'algoritmo sfrutta la tecnica del "Divide et impera": suddivide una lista non ordinata in due sottoliste (ricorsivamente), le ordina separatamente e quindi le unisce per ottenere una lista ordinata. È noto per la sua stabilità e la sua efficienza nell'ordinamento di grandi quantità di dati.
- **Determinante matrice**: data in input una matrice, viene calcolato il determinante utilizzando l'algoritmo di Laplace. Quest'ultimo, in modo ricorsivo, calcola i determinanti di tutte le sottomatrici a partire dagli elementi di una riga e, dopo aver moltiplicato il determinante per l'elemento scelto, li somma. Il passo base si ha quando la matrice è 1x1.

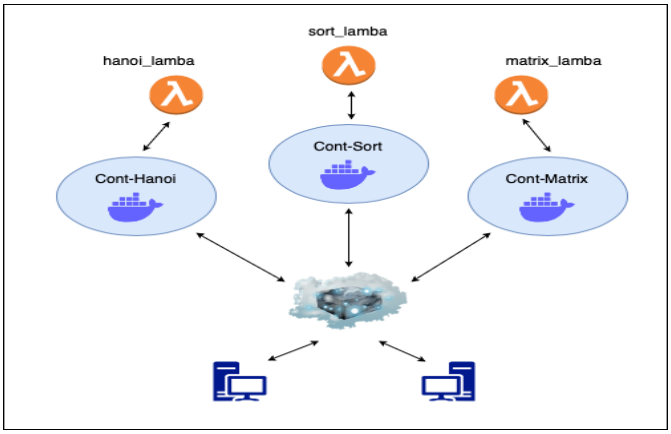

Fig. 2. Architettura dell'applicazione

Ogni funzione viene eseguita dal server all'interno di un container diverso: avremo cosi 3 container. Questi sono stati sviluppati utilizzando Docker, attraverso le API Docker di Python, e montano un'immagine di base Python 3.9. Essendo i container isolati tra loro e dal sistema sottostante, in fase di costruzione, vengono copiati in essi i file necessari all'esecuzione della funzione e vengono installate le dipendenze necessarie. Quando la funzione viene chiamata, se il container è già attivo viene subito eseguita, altrimenti il container viene creato e poi verrà svolta la funzione. A seconda di alcune politiche che variano da funzione a funzione, l'esecuzione può essere trasferita, totalmente o in parte, su AWS Lambda: attraverso l'offloading è possibile diminuire notevolmente il carico di lavoro della CPU e i tempi di risposta, soprattutto in caso di input grandi e, di conseguenza, di esecuzioni computazionalmente onerose. L'applicazione è infatti pensata per poter girare anche su dispositivi con poca capacità computazionale, servendosi appunto di un server e del servizio offerto da Amazon Web Service.

IV. Implementazione

Di seguito viene illustrata l'implementazione di ciascuna funzione e le scelte adottate per effettuare l'offloading.

A. Torre di Hanoi

Questa funzione è stata scelta perché, essendo un problema matematico risolvibile attraverso la ricorsione, per un numero di dischi abbastanza grande, lo svolgimento, e quindi l'ottenimento di tutti i passi necessari alla risoluzione, è computazionalmente oneroso. Questo può portare problemi, quali aumento esponenziale dei tempi di risposta o impossibiltà di risolvere la ricorsione, in dispositivi con caratteristiche hardware non eccellenti.
La risoluzione consiste nello spostare inizialmente gli n-1 dischi dal piolo iniziale a quello di d'appoggio chiamando ricorsivamente la funzione. Il passo base si ha quando il numero dei dischi è 1 (n==1), e in questo caso viene ritornato il passo necessario per spostare quel disco dal piolo di partenza a quello finale. Successivamente gli n-1 dischi verranno spostati, sempre chiamando la funzione in modo ricorsivo, dal piolo d'appoggio a quello di destinazione, passando ovviamente per il passo base. È stato dimostrato matematicamente che il numero minimo di mosse necessarie per spostare correttamente i dischi è pari a $2^n - 1$. La funzione ritornerà una lista con elementi del tipo [1 A C], dove il numero indica il disco, A il piolo di partenza e C quello di destinazione. All'interno del container vengono svolte tutte le esecuzioni con un numero di dischi in input minore di 4, per rendere appunto la funzione eseguibile anche su dispositivi (o server) più semplici, mentre per un numero maggiore di 4 viene sfruttato il servizio AWS Lambda attraverso la libreria boto3 di python e l'API di Lambda per invocare la funzione "hanoi_lambda". L'input alla funzione verrà passato attraverso un elemento JSON ed anche il risulato verrà ritornato in un elemento JSON dal quale poi verrà estratta la lista finale. Dopo aver ottenuto il risultato, il server invierà il risultato al client come risposta alla chiamata http precedentemente effettuata e verrà visualizzato nell'apposito campo risultato della GUI.

B. Merge Sort

Questa seconda funzione, ovvero l'ordinamento attraverso l'algoritmo Merge Sort, è stat scelta principalmente perché, da costruzione, il suo punto focale è la divisione (anche essa ricorsiva) della sequenza di numeri in due metà e l'ordinamento di esse separatamente: ciò si presta bene all'implementazione di una politica di offloading che prevede di trasferire una parte della computazione su AWS, diminuendo cosi l'utilizzo della CPU e il tempo di risposta per sequenze numeriche molto lunghe.
La funzione prende in input la lista inserita dall'utente nell'apposito campo della finestra dedicata nella GUI e, dopo aver trovato il punto medio di essa, la divide in due parti e ognuna delle due parti viene passata ricorsivamente alla funzione iniziale. Il passo base si ha quando la lunghezza della lista è uguale ad 1, perché ovviamente una lista composta da un solo elemento è di per se ordinata. Successivamente

le due metà ordinate vengono combinate tra loro in modo iterativo, attraverso dei cicli che scorrono entrambe le liste. All'interno del container vengono svolti in modo integrale tutti gli ordinamenti di liste con lunghezza minore o uguale a 10, mentre, per le liste di lunghezze maggiore, viene combinato l'utilizzo locale con l'offloading su AWS. Nel caso di liste lunghe, l'esecuzione dei vari rami di ricorsione potrebbe risultare lunga e consumare una quantità di CPU notevole, per questo si è deciso di far eseguire l'ordinamento ricorsivo dei primi 10 numeri in locale, mentre la restante parte verrà ordinata tramite una richiesta ad AWS Lambda, che ordinerà la lista e la restituirà al container. Questo limite può essere configurato nel file di configurazione, attraverso il paramentro "number_offload_sort": viene però impostato a 10 come valore di partenza. Successivamente, le due liste, ordinate su piattaforme differenti, verranno combinate in locale e restituite all'utente sotto forma di risultato. Anche in questa funzione il passaggio di valori di input e risultati avviene attraverso elementi JSON, in funzione di chiamate http.

C. Determinante Matrice

Per la terza funzione la scelta è ricaduta sul calcolo del determinante di una matrice attraverso l'algoritmo di Laplace, che sfrutta una strategia ricorsiva per ottenere il risultato desiderato. Questo metodo parte da casi base: se la matrice è 1x1, il determinante è l'elemento stesso, mentre se è 2x2, il determinante può essere calcolato attraverso la formula $ad - bc$. Quando abbiamo matrici di dimensioni maggiore, il processo diventa più complicato: si parte scegliendo una riga (o colonna) e per ciascun elemento di quella riga (o colonna) si calcola il determinante (che in questo caso assume in nome di "cofattore") di una sottomatrice ottenuta eliminando la riga e la colonna a cui appartiene l'elemento scelto. Qui il calcolo del determinante segue lo stesso processo e si entra nella fase ricorsiva, ottenendo risultati parziali fino a che non avremo matrici 1x1 o 2x2. Da qui potremo calcolare i vari determinanti delle sottomatrici e sommandoli otterremo il determinante della matrice data in input.
Per matrici di dimensioni grandi questo algoritmo risulta computazionalmente oneroso, per cui si è deciso di trovare le sottomatrici della matrice originale in locale e calcolare i cofattori (ovvero i determinanti delle sottomatrici) parallelamente sfruttando AWS Lambda. In questo modo la computazione più onerosa viene trasferita, oltretutto parallelizzandola, su AWS, lasciando in locale soltanto la moltiplicazione tra l'elemento e il relativo cofattore e la somma finale. Questo permette di poter eseguire la funzione anche con poca potenza computazionale, lasciando gran parte del lavoro ad AWS. Proprio nella funzione lambda "matrix_lambda", viene calcolato il determinante della sottomatrice ricevuta in input, avvelendosi della libreria Python "numpy". Va specificato che, per importare questa libreria su AWS, è stato necessario l'utilizzo di un "Layer", ovvero un pacchetto di librerie, dipendenze o codice personalizzato che può essere condiviso tra diverse funzioni AWS Lambda. Nel caso specifico è stato utilizzato il layer "AWSSDKPandas-Python311", compatibile con Python 3.11 e architetture x86-64. Anche in questo caso vengono utilizzate richieste/risposte http e passaggio dell'input/risultato sotto forma di elementi JSON.

D. Container

I container, all'interno dell'applicazione, sono sviluppati attraverso Docker ed in particolare le Docker API di Python. Sono stati scritti innanzitutto i tre Dockerfile, necessari per la build di un'immagine docker: in ognuno troviamo le istruzioni di copy dei file necessari all'esecuzione della funzione, la creazione con mkdir di una cartella root/.aws, e le copy dei file credentials e config di AWS per l'offloading. Inoltre troviamo l'installazione delle dipendenze flask e boto3. A partire da questi dockerfile, quando richiesto, vengono create le immagini e, a partire da queste, vengono runnati i container. Merita una menzione anche il "ports mapping": per permettere ai container di parlare con il mondo circostante, vengono mappate delle porte di esso con delle porte del server, con possibilità di specificare anche il protocollo di comunicazione (nel nostro caso tcp). Per mantenere la decentralizzazione dell'applicazione, questa gestione spetta al server. È stato implementato inoltre anche un meccanismo di "garbage collector": il server, tramite un nuovo thread sempre attivo, ottiene, per ogni container in stato "exited", l'ultimo istante di accesso a quel determinanto container e, se viene rilevato che quest'ultimo non è stato acceduto negli ultimi due minuti, il container selezionato viene rimosso. Il thread viene messo in pausa per 120 secondi, al termine dei quali viene attivata la funzione di controllo, e successivamente rimesso in pausa. Inoltre, quando viene chiusa la GUI, vengono rimossi, se presenti, tutti i container in esecuzione.

E. Configurazione

Prima di far partire il server e l'applicazione, è necessaria la configurazione di alcuni valori all'interno del file config.json. Nel file troviamo:

- **Porte**: sono rispettivamente la porta che utilizza il client per parlare con il server, e le tre porte che utilizza il server per parlare con i tre container dedicati alle funzioni.
- **Numero offload sort**: è il numero che rappresenta il limite di elementi ordinabili solo in locale; per liste di lunghezza maggiore, verrà effettuato anche l'offloading.
- **Url Server**: rappresenta l'URL del server, con il valore xxx che verrà sostituito in base alla porta scelta precedentemente; Nel caso base, può essere lasciato "localhost" per far eseguire il server sulla macchina locale.
- **Credenziali AWS**: come specificato nella sezione I, per poter effettuare correttamente l'offloading su AWS Lambda e quindi eseguire le funzioni desiderate, è necessario inserire all'interno di questo file le chiavi "aws_access_key_id" e "aws_secret_access_key" ed il token "aws_session_token". Questi valori possono essere presi dal Learner Lab di AWS ed hanno una validità di 4 ore, dopo le quali vanno rigenerati.

V. Limitazioni riscontrate

A. SPOF

In caso di utilizzo del sistema con un solo server, come nel nostro caso base, questo può rappresentare un ”Single Point Of Failure”: è possibile risolvere ciò aumentando il numero di server, ma è necessario ogni volta modificare la ”server port” nel file di configurazione config.json, altrimenti verrà restituito un errore a causa dell'utilizzo simultaneo di una singola porta.

B. Cold Start

La prima volta che viene eseguita una funzione, ci si trova inevitabilmente davanti al fenomeno del ”Cold Start”: è necessario un certo tempo tecnico per creare il container e inizializzare l'ambiente di esecuzione. Questo problema non si ripresenta se la funzione viene eseguita nuovamente nel giro di un tempo breve (2 minuti), perché il container viene fermato ma non viene rimosso, ed è pronto all'esecuzione. Dopo i due minuti, come specificato, inteverrà il ”garbage collector” per salvaguardare memoria e utilizzo della CPU.

C. Persistenza

Data l'assenza di ”Volumes Docker”, e l'utilizzo di chiamate HTTP sincrone per scelta implementativa, tutti i risultati ottenuti verranno persi quando verrà eseguita nuovamente la funzione. Se necessario, in futuro, è possibile modificare l'applicazione e prevedere il salvataggio dei risultati in un file locale oppure su Amazon s3.

D. Credenziali

Come già specificato sopra, le credenziali AWS hanno una validità di 4 ore, dopodichè va fatto ripartire nuovamente il laboratorio AWS e vanno prese le nuove chiavi e il nuovo token: viene da sé che non è possibile usare consecutivamente l'applicazione per più di 4 ore.