Progeto B3: Funzioni serverless nel continuo cloud-edge

ALESSANDRO COSTA - 0300040

ABSTRACT Lo scopo del progetto è realizzare in linguaggio di programmazione Python un'applicazione distribuita che implementi l'esecuzione di funzioni serverless in ambito edge e in ambito cloud. L'obiettivo dell'applicazione è permettere ad un server che opera in uno scenario di edge computing di decidere se eseguire localmente la funzione richiesta dagli utenti oppure di effettuare l'offloading ad un provider Cloud che offre un servizio di serverless computing, con lo scopo finale di migliorare la qualità di servizio sperimentato dagli utenti.

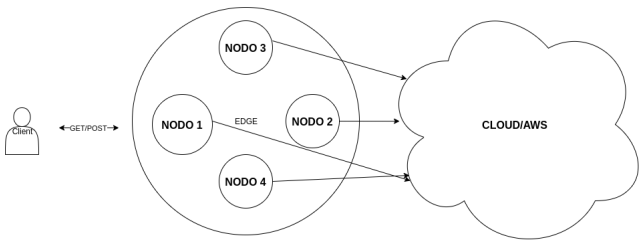
I. INTRODUZIONE

Lo scopo del progetto è di realizzare in Python un'applicazione distribuita che implementi funzioni serverless in ambito edge e in ambito cloud. In grandi linee, questa applicazione prende come input quale operazione deve essere effettuata delle due e il valore intero associato per il calcolo dell'operazione. Le operazioni scelte in questione sono il calcolo della precisione del pi-greco e il calcolo dell'ennesimo numero primo, entrambe funzioni CPU-intensive. Sia l'applicazione client che quella dei diversi nodi edge sono state inserite in un container docker. Il fine ultimo dell'applicazione è quello di diminuire il tempo di risposta della richiesta e di non sovraccaricare il nodo edge preso in causa, utilizzando infatti la funzionalità di AWS lambda nella necessità.

II. SCELTE ARCHITETTURALI E PROGETTUALI

L'ambiente emulato nel progetto è composto da 3 tipi di componenti:

1) Client: è stato simulato un terminale dove verrà fatto partire un programma in python chiamato "client.py" che avrà il compito di interagire con l'utente per effettuare la richiesta a uno dei nodi edge.
2) Nodo Edge: è il componente principale del sistema che sarà sempre in ascolto poichè in attesa delle richieste dell'utente. Oltre a sviluppare il calcolo delle due funzioni serverless, sceglie se sia opportuno continuare il calcolo in locale o di demandare la richiesta a un servizio esterno Lambda.
3) AWS lambda: servizio cloud chiamato dai nodi edge quando ritengono opportuno trasferire il calcolo computazionale, per evitare l'uso intensivo della CPU del dispositivo, che in uno scenario reale potrebbe servire per fare altri calcoli computazionali.



A. CLIENT - GESTIONE DELLE RICHIESTE UTENTE

Il Client è stato progettato per interfacciare l'utente con l'applicazione, gestendo l'input e trasferendo la richiesta di calcolo ai nodi edge. Sarà sempre in ascolto, in attesa che l'utente inserisca i giusti valori di input. Suddivisa in due funzioni principali:

1) Controllo(): racchiusa da un ciclo while infinito per garantire l'ascolto continuo delle richieste degli utenti, cercando di catturare gli input in modo corretto, implementando oltre la richiesta di input delle due funzioni, la possibilità di leggere 2 file dove al loro contenuto verrà salvato l'ultimo risultato calcolato e i tempi dell'esecuzione dell'ultimo processo chiamato. A inizio funzione prendo un numero random tra 3 e 4 poichè simulo che nel sistema ci siano 2 nodi edge, ai quali passerò la richiesta. La scelta della randomicità è per semplicità, poiche questo tipo di controllo sia necessario nel caso in cui si conosca la locazione geografica di tale nodi. Il valore random tra 3 e 4 è dato dal fatto che per comunicare i container docker sia necessario conoscere il loro indirizzo IP, che per default i due nodi edge verranno asseggati al numero 3 e al numero 4.
2) func(operazione,value,i-esimo nodo edge): una volta presi i valori di input, si crea il link per comunicare con

il nodo edge scelto in precedenza, chiamandolo tramite comando curl. Una volta mandata la richiesta, il client simula l'invio della richiesta aspettando un secondo, per poi essere di nuovo operativo e pronto a prendere le nuove richieste da input.

B. DOCKER FILE - IMAGE - CONTAINER PER CLIENT

Tutte le applicazioni Python create, sono state assegnate ai vari 'oggetti' definiti nello schema architetturale. Nel caso Client, il docker file racchiude pochi comandi, in quanto sarà necessario installare le librerie Python e lanciare il Container come shell, per permettere all'utente di accedere alla funzione che sarà pronta per prendere i diversi input.

C. EDGE - CALCOLO CPU INTENSIVE FUNCTIONS E GESTIONE DEL CLOUD

Il codice sorgente dei nodi Edge ha la funzionalità di calcolare una delle due funzioni CPU intensive e in caso di demandare la richiesta a un servizio cloud esterno. Il calcolo della funzione del pi-greco è basata sulla serie di Eulero, invece per il calcolo del numero primo e il vettore dei numeri primi fino all'ultimo richiesto è stato fatto da 0. Per un esecuzione pulita si è pensato di controllare l'utilizzo della CPU del container, affinchè se superata una soglia limite, si affida l'esecuzione al servizio lambda e rimane in attesa. Questo è pensato per permettere all'utente di poter continuare a utilizzare il proprio device senza attendere la fine dell'esecuzione. Per l'invocazione del servizio Lambda mi sono affidato alla libreria boto3. La gestione delle credenziali per poter eseguire il servizio Lambda in caso di invocazione è gestito tramite il comando di avvio del container per i nodi Edge. Questo implica che nel locale sarà necessario possedere il file /.aws/credentials con l'ID, PASSWORD e TOKEN corretti.

D. DOCKER FILE - IMAGE - CONTAINER PER EDGE

La creazione dell'imagine del container per i nodi Edge è simile alla precedente, con la differenza che dovrà essere lanciata in modalità "demone", cosi da essere sempre in ascolto per qualsiasi tipo di richiesta giunta dal Client.

III. LIMITAZIONI PROGETTUALI

Sono state riscontrate diverse limitazioni, tra le quali si ritiene siano le più importanti le seguenti:

- La gestione delle credenziali: sarebbe necessario catturare ogni volta in modo autonomo le credenziali dal pc locale da passare ai container dei nodi edge, cosi che possano demandare la richiesta al servizio lambda quando necessario. Questa è una limitazione notevole in quanto la gestione del servizio dovrebbe essere garantita del fornitore dell'applicazione e non dall'utente.
- Scelta del nodo Edge da invocare: in uno scenario realistico, i vari nodi Edge sono sparsi per il mondo e l'invio di una richiesta di calcolo andrebbe fattà in base alla distanza e la sua posizione geografica. Ovviamente queste componenti andranno a impattare notevolmente sul tempo di esecuzione dell'applicazione, e per semplicità progettuale si è scelto di prendere un campione minimo di nodi Edge e mandare la richiesta in modalità random tra i nodi.
- Controllo delle risorse: la gestione del calcolo delle funzioni è garantito da un controllo sulla CPU che se supera una certa soglia(80perc di utilizzo) manda la richiesta al servizio Lambda. Adottando questa politica, la maggior parte dei valori possibili verranno inoltrati al servizio di AWS. Questo garantisce al Client di poter utilizzare il proprio dispositivo senza processi in esecuzione, ma dovendo aspettare diverso tempo per ottenere la risposta del calcolo.

IV. TESTING

Per testare l'efficienza, la prontezza e la robustezza del sistema ho provato in modo empirico tre carichi diversi di input, che passano da un carico leggere a uno pesante. Essendo due tipo di funzioni differente con processo di calcolo differente, ho cercato 2 input piccoli, 2 medi e 2 pesanti. Per il calcolo della precisione del pi-greco ho preso in considerazione:

- Carico Leggero Locale : 1000000 - quasi istantaneo, circa 2 secondi;
- Carico Leggero Aws: 1000000 - circa 10 secondi.
- Carico Medio Locale: 2000000 - circa 6 secondi;
- Carico Medio Aws: 2000000 - circa 16 secondi;
- Carico Pesante: 7000000 - circa 11 secondi;
- Carico Pesante Aws: 7000000 - circa 50 secondi;

Per il calcolo del'ennesimo numero primo e il vettorei dei numeri primi precedenti ho preso in considerazione:

- Carico Leggero Locale: 100- istantaneo, qualche millisecondo;
- Carico Leggero Aws: 100- circa 1 secondo;
- Carico Medio Locale: 1000 - circa 1 secondo;
- Carico Medio Aws: 1000 - circa 11 secondi;
- Carico Pesante Locale: 2500 - circa 5 secondi;
- Carico Pesante Aws: 2500 - circa 42 secondi;

Come si può notare dai dati vi è una differenza notevole dei tempi, e stranamente il tempo di calcolo in remoto è di gran lunga maggiore. Ritengo che questo risultato sia tale poichè quando ho gestito il servizio lambda, anche esso è stato sviluppato in un container che però viene gestito da Lambda. Probabilmente il container verrà lanciato con i comandi base senza un particolare numero di CPUs che garantiscono una maggiore velocità del servizio. Oltre a ciò dobbiamo sommare problemi di latenza e comunicazione internet.

V. AMBIENTE DI SVILUPPO

L'applicazione è stata sviluppata sul sistema operativo Linux Ubuntu, sul programma PyCharm per avere agevolazioni di scrittura durante la fase di coding. Sono state utilizzate diverse librerie, le quali sono state trasferite anche ai relativi container per permettere l'esecuzione delle applicazioni .py. Le librerie fondamentali utilizzate sono:

- subprocess: necessaria per chiamare il comando curl, per l'invocazione dei nodi Edge, grazie ad essa sono riuscito anche a salvarmi su due file il risultato delle operazioni e il tempo di esecuzione.
- math: utilizzata per il calcolo della funzione del pi-greco.
- psutil: necessaria per il controllo della gestione delle risorse, in quanto se la CPU del nodo superava una certa soglia, la richiesta del calcola era indirizzata verso il servizio cloud.
- boto3: libreria per invocare il servizio AWS.

VI. CONCLUSIONI

In conclusione si può evincere come il processo di esecuzione gestito da un ente esterno, come AWS possa rendere più gestibile il device nel momento in cui possa essere occupato da calcoli pesanti di CPU che potrebbero portare il device a rimanere bloccato per calcoli di complessità elevata. Il tempo di risposta effettivo non viene decrementato per diversi fattori:

1) La latenza per la connessione al sercizio;
2) La distanza tra i nodi;
3) La gestione del container in Lambda.