Sistemi Distribuiti e Cloud Computing: FaaS Management

Fardella Roberto - 0334186

16 Ottobre 2023

Fig. 1. logo università

1 INTRODUZIONE

Function-as-a-Service (FaaS) è un tipo di servizio di cloud computing che consente agli sviluppatori di creare, eseguire e gestire pacchetti applicativi come funzioni, senza la necessità di mantenere un'infrastruttura dedicata. FaaS è un modello di esecuzione basato su eventi, che opera all'interno di container stateless. Le funzioni sono progettate per gestire la logica e lo stato lato server, sfruttando i servizi offerti dai provider FaaS. Le ragioni principali per lo sviluppo di un FaaS management includono la scalabilità dinamica, il bilanciamento del carico di lavoro sulle risorse del nodo locale ed il servizio di calcolo cloud . In questo rapporto, creeremo un sistema di gestione di FaaS personalizzato per bilanciare il carico tra risorse locali e AWS Lambda,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. Inoltre, esamineremo tutti gli aspetti cruciali del progetto, dalla progettazione alla sua implementazione.

2 DESCRIZIONE DELL'ARCHITETTURA

Il sistema è dato da tre macro componenti (figura 2):

- **Un client** in cui gli utenti si possono interfacciare per poter invocare 3 possibili funzioni a loro scelta ;
- Una piattaforma di virtualizzazione leggera, chiamata **Docker**, che consente di creare, distribuire e gestire le funzioni all'interno dei container;
- **AWS Lambda**, una piattaforma di calcolo serverless event-driven fornita da Amazon come parte di Amazon Web Services.

Nel sistema, oltre a questi tre componenti, abbiamo un **gestore delle container e dei metadati dell'applicazione**, che si occupa ad esempio, del supporto nell'implementazione della politica di offloading e della gestione di esecuzione dei container, tenendo in considerazione:

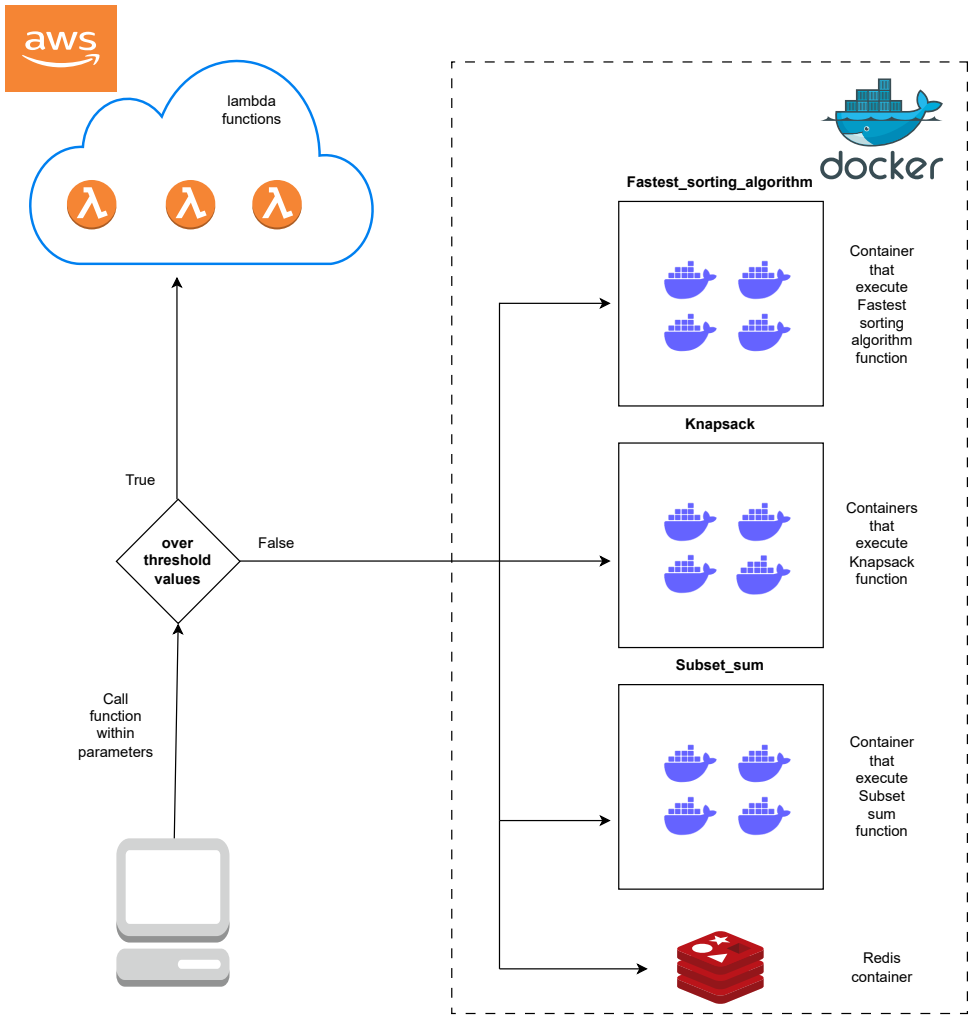

Fig. 2. Architettura del FaaS management

- Il problema del cold start;
- Chiamate simultanee della stessa funzione.

2.1 Funzioni

Le funzioni sono state scritte in **linguaggio Go.**
In base ad una politica di offloading, esse possono essere eseguite in:

- Locale, distribuite in dei container Docker;
- Remoto, Esecuzione di funzioni Lambda, distribuite attraverso dei file zip.

2.1.1 ***Fastest sorting algorithm.*** Si propone di confrontare tre algoritmi di ordinamento molto noti: "Bubble Sort," "Selection Sort," e "Merge Sort." L'obiettivo è determinare quale di questi algoritmi è il più veloce per ordinare un array di numeri interi. Per fare ciò, abbiamo implementato una funzione chiamata findFastestSortingAlgorithm.

2.1.2 ***Knapsack.*** Il "Problema dello Zaino" (in inglese, "Knapsack Problem") è un classico problema di ottimizzazione combinato. In questo problema, un selezionatore deve decidere quali oggetti scegliere da un insieme di oggetti, ognuno con un valore specifico e un peso specifico, in modo da massimizzare il valore totale degli oggetti selezionati senza superare un limite di peso prestabilito.

2.1.3 ***Subset sum.*** Il "Subset Sum Problem" è un altro problema di ottimizzazione combinatoria in cui l'obiettivo è determinare se esiste un sottoinsieme di elementi da un insieme dato tale che la somma dei valori di questi elementi sia uguale a un valore di destinazione specifico. In altre parole, il problema consiste nel trovare un sottoinsieme degli elementi in modo che la loro somma sia uguale a un valore obiettivo desiderato.

2.2 Politica di offloading

La politica di offloading **threshold-based** è basata su un set di valori di threshold, che vengono prelevati da un file di configurazione in formato JSON, chiamato "config.json".
Il file JSON viene acceduto periodicamente da un thread a Runtime, che è quindi possibile cambiarne i valori rendendo il FaaS Management più flessibile e cercando di ridurre il grado di application-dependency della politica di offloading adottata, a seconda delle esigenze necessarie che ne derivano dal contesto di utilizzo applicativo.

I valori di threshold sono utilizzati per determinare quando una funzione serverless deve fare l'offloading sul servizio cloud AWS Lambda. Ad esempio, una serie di criteri che sono stati utillizzati per fare l'offloading si basano sulle risorse del sistema all'interno dell'environment Docker . Un thread a Runtime utilizza i valori di threshold per monitorare le funzioni serverless in esecuzione, confrontandole con le metriche raccolte e collezionate periodicamente nella Redis cache.
Quando un valore di threshold viene superato, il thread che si occupa di impostare dove verrà eseguita la funzione, setterà una flag (per permettere l'offloanding delle funzioni nel cloud) al valore True, facendo cosi il redirect delle prossime richieste di esecuzione.

2.3 Cold start

Il cold start si verifica quando l'utente invia una richiesta di invocazione di una certa funzione e non è disponibile nessun container, richiedendo quindi una latenza aggiuntiva al tempo di risposta, relativa alla fase di startup del container.
Per affrontare questo problema, abbiamo implementato una soluzione che mantiene il container inattivo per un determinato periodo di tempo, prima di rimuoverlo. Tale periodo di tempo che rappresenta un intervallo di inattività del container, configurabile nel codice della applicazione, un

thread prende la lista dei container inattivi e ne salva il timestamp in cui li trova inattivi, se non presenti.
Se tale timestamp è maggiore dell'intervallo temporale, allora il container verrà rimosso.

3 IMPLEMENTAZIONE & TECNOLOGIE UTILIZZATE

3.1 Docker

Il deployment delle funzioni (scritte in linguaggio Go) e stato fatto, in parte, attraverso la piattaforma docker.

3.1.1 ***Dockerfile.*** l'immagine che verrà poi utilizzata per creare un container, viene costruita tramite un **multistage Dockerfile**. Un multistage Dockerfile è un Dockerfile che utilizza almeno una immagine di base per costruire l'immagine finale. Questo può essere utile per ridurre la dimensione dell'immagine finale e per migliorare la sicurezza del Dockerfile.

La prima immagine è un'immagine ufficiale di **Go**, utilizzata poi come immagine di base per costruire quella finale chiamata **base-debian11.**

3.1.2 ***Utilizzo di Redis.*** Nel contesto del nostro progetto, abbiamo identificato diverse esigenze cruciali per garantire le prestazioni, la scalabilità e l'affidabilità del nostro sistema. Una componente chiave nella realizzazione di tali obiettivi è stata l'implementazione di Redis, un sistema di memorizzazione dati in memoria altamente performante. Di seguito, discuteremo in dettaglio come abbiamo utilizzato Redis in due importanti aspetti del nostro progetto.

- **Broker di Messaggi Pub/Sub:** Redis è noto per le sue capacità di messaggistica pub/sub. È utile per la propagazione dei parametri ai container in un'architettura serverless o FaaS.
  è possibile pubblicare i messaggi contenenti i parametri e sottoscrivere i container o le funzioni interessate a tali messaggi. Questo consente una comunicazione asincrona tra le diverse parti del sistema.

- **Sistema di Cache:** Consente un accesso rapido e riduce il carico sui sistemi di memorizzazione permanente. Questo è particolarmente utile per il monitoraggio delle prestazioni e per l'ottimizzazione dei container. In particolare,è stato utilizzato Redis come sistema di cache per i seguenti scopi:

  - **Supporto al Cold Start:** riduce i tempi di avvio dei container. Un container resta inattivo per un periodo di tempo predefinito. Questo periodo di tempo viene quindi confrontato con il timestamp del container nel momento in cui passa allo stato "exited", ossia quando termina l'esecuzione della funzione;

  - **memorizzazione delle metriche di sistema:** è stato impiegato come sistema di cache per memorizzare le metriche di sistema in tempo reale, evitando l'utilizzo di un database tradizionale e permettendo l'accesso rapido ai dati durante l'esecuzione delle nostre applicazioni serverless, riducendo inoltre il carico sul sistema. Le misurazioni effettuate verranno successivamente utilizzate nella politica di offloading (threashold-based) implementata;

3.1.3 ***Librerie.***

- La libreria **boto3**, kit di sviluppo del software (SDK) Amazon Web Services (AWS) per Python. Boto3 consente agli sviluppatori Python di scrivere software che utilizza i servizi AWS;

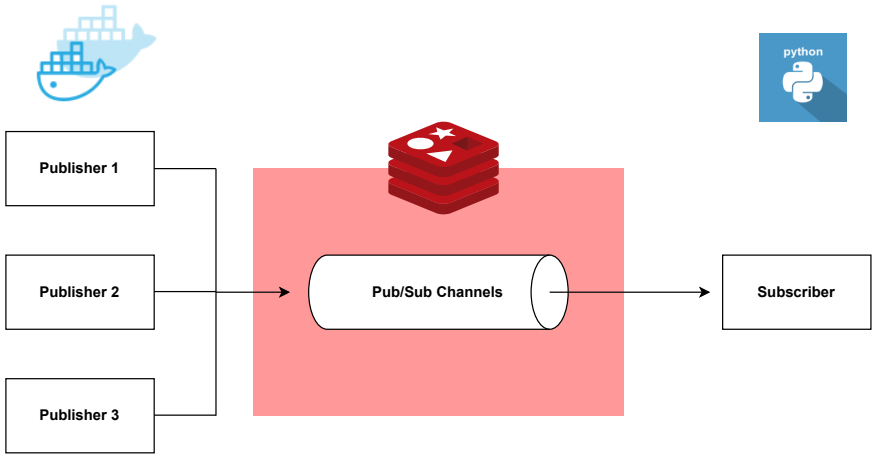

Fig. 3. valori di ritorno restituiti tramite pattern pub/sub

- Il client è stato sviluppato in Python con la libreria di interfaccia grafica **TKintker**
- **Docker-py**, libreria Python per l'interazione con l'API Docker Engine. Docker Engine è un software che consente di creare, gestire e distribuire container
- La libreria Redis per Python, **redis-py**. Mentre la libreria Redis per Go è **go-redisV7**.

3.1.4 ***AWS Lambda***. AWS Lambda è un servizio serverless che consente di eseguire codice senza dover gestire server o infrastrutture. Le funzioni Lambda sono progettate per essere eseguite rapidamente e in modo efficiente, e scadono automaticamente dopo un timeout specificato (nel laboratorio AWS, il timeout settato per l'applicazione è di 15 secondi).

L'utilizzo di AWS Lambda per il bilanciamento del carico presenta i seguenti vantaggi:

- **Riduce il carico sull'environment Docker**. Lambda può essere utilizzato per eseguire le funzioni che richiedono molto tempo o risorse, in modo da allegerire le risorse del server;
- **Migliora le prestazioni dell'applicazione**. Lambda può essere utilizzato per eseguire le funzioni in modo rapido e efficiente, migliorando le prestazioni dell'applicazione.
- **garatisce scalabilità**: è possibile invocare un numero illimitato di funzioni purchè il servizio sia disponibile.

4 CONFIGURAZIONE

4.1 file di configurazione json

separare i dati dal codice utilizzando un file JSON è un modo efficace per migliorare la manutenibilità, la flessibilità, la trasferibilità e la scalabilità del sistema. è stato utilizzato un **file di configurazione JSON** suddiviso in quattro categorie chiave:

- **Configurazione dei Dockerfile:** percorsi dei Dockerfile per ciascun componente del sistema;

- **Configurazione del Server Redis:** contiene l'indirizzo e la porta del server Redis utilizzato nel progetto;
- **Valori di Threshold;**
- **Canali per la sottoscrizione e pubblicazione (pub/sub) utilizzati per la comunicazione tra i componenti del sistema.**

4.2 Credenziali AWS Lambda

Per poter effettuare correttamente l'offloading su aws Lambda, va configurato inizialmente il file delle credenziali .aws/credentials.
Le credenziali AWS hanno una validita di 4 ore, dopodiche 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 piu di 4 ore.