Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica per il Management

ANALISI COMPARATIVA DI SOLUZIONI SERVERLESS

Relatore:
Chiar.mo Prof.
Rossi Davide

Presentata da:
De Rosa Davide

II Sessione
Anno Accademico 2023/2024

(DA FARE ALLA FINE)
5 parole chiave per caratterizzare il contenuto della dissertazione:
(se non ti piacciono così sparse puoi anche semplicamente scriverle su una riga sola)

parola 5

parola 4

parola 3

parola 2

Parola 1

Abstract

abstract

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

da fare alla fine

1.1 Scopo della Tesi

Introduzione agli obiettivi della tesi, come il confronto tra soluzioni serverless e l'analisi di AWS e Google Cloud.

1.2 Metodologia

Descrizione dell'approccio adottato per l'analisi e il confronto delle due piattaforme.

1.3 Struttura della Tesi

Breve descrizione dei capitoli successivi.

Capitolo 2

Nozioni di base su Serverless

2.1 Definizione e Concetti Fondamentali

Il *Serverless Computing* è una tecnologia in rapida crescita che sta avendo un impatto sempre più significativo sulla società, trovando ampia adozione sia nel mondo accademico che in quello industriale. La sua promessa principale è rendere i servizi informatici più accessibili, personalizzabili in base alle esigenze specifiche e a basso costo, delegando all'infrastruttura la gestione dei problemi operativi.

I principali fornitori di servizi cloud, come *Amazon*, *Microsoft*, *Google* e *IBM*, offrono piattaforme serverless già pronte all'uso, con ben definite responsabilità e prezzi.

Un sistema può essere considerato serverless se presenta le seguenti caratteristiche:

- **Auto-scaling**. La capacità di adattarsi automaticamente alle variazioni del carico di lavoro, scalando sia orizzontalmente che verticalmente, è un elemento chiave. Un'applicazione serverless può ridurre il numero di istanze fino a zero, introducendo il concetto di *cold startup*, che può causare ritardi nel tempo di risposta dovuti alla necessità di avviare l'ambiente da zero e caricare il codice.
- **Pianificazione flessibile**. Non essendo vincolata a un server specifico, l'applicazione viene pianificata dinamicamente in base alle risorse disponibili nel cluster, garantendo bilanciamento del carico e prestazioni ottimali. Inoltre, la pianificazione tiene conto di più regioni geografiche per evitare interruzioni del servizio in caso di malfunzionamenti o crash.
- **Event-driven**. Le applicazioni serverless si attivano in risposta a eventi, come richieste HTTP, aggiornamenti di code di messaggi o nuove scritture su servizi di storage. Associando trigger e regole agli eventi, il sistema può rispondere

in modo efficiente e flessibile alle diverse tipologie di input. Gli eventi possono essere attivati non solo da fonti esterne alla piattaforma cloud, ma anche internamente, attraverso i vari servizi offerti dalla piattaforma stessa. Questo permette agli sviluppatori di creare applicazioni distribuite che utilizzano diversi servizi cloud in modo integrato. Il serverless computing rappresenta una parziale realizzazione di un modello basato sugli eventi, dove le applicazioni vengono definite dalle azioni e dagli eventi che le attivano. Questo concetto richiama i sistemi di database attivi e riflette la letteratura sui sistemi event-driven, che da tempo teorizza l'esistenza di sistemi informatici generali in cui le azioni sono elaborate in modo reattivo ai flussi di eventi.

Le piattaforme serverless basate su funzioni adottano pienamente questa visione, utilizzando astrazioni semplici come le funzioni per definire le azioni e costruendo la logica di elaborazione degli eventi direttamente all'interno del cloud. In questo modo, il serverless computing offre un framework flessibile per la gestione e l'elaborazione degli eventi su larga scala.

- **Sviluppo trasparente**. Gli sviluppatori non devono più preoccuparsi della gestione delle risorse fisiche o dell'ambiente di esecuzione, poiché queste responsabilità sono delegate ai provider cloud. Questi ultimi si occupano di garantire la disponibilità delle risorse fisiche, la sicurezza e la potenza di calcolo, rendendo tutto ciò trasparente agli sviluppatori, facilitando così il processo di sviluppo e distribuzione.
- **Pagamento in base al consumo**. Il serverless trasforma il costo della capacità di calcolo da una spesa di capitale a una spesa operativa, eliminando la necessità per gli utenti di acquistare server dedicati per i picchi di carico. Il modello *pay-as-you-go* permette di pagare solo per le risorse effettivamente utilizzate.

Un modello di calcolo che soddisfa queste cinque caratteristiche è considerato serverless. La sua crescente diffusione è dovuta in parte al fatto che gli sviluppatori possono pagare solo in base all'uso effettivo delle risorse, piuttosto che per una capacità preallocata.

Oggi, il serverless computing viene utilizzato principalmente in scenari backend per lavori batch, come l'analisi dei dati, attività di machine learning e applicazioni web basate su eventi.

2.2 Funzioni Serverless

La modalità tradizionale di distribuzione nell'ambito dell'*Infrastructure-as-a-Service* (*IaaS*) richiede che un server sia attivo a lungo termine per garantire servizi continui. Tuttavia, questa allocazione esclusiva implica che le risorse vengano mantenute anche quando l'applicazione non è in esecuzione. Di conseguenza, l'utilizzo delle risorse nei data center è generalmente basso, attestandosi in media intorno al 10%, specialmente per i servizi online con un uso prevalentemente diurno. Questa inefficienza ha portato allo sviluppo di un modello di servizio on-demand gestito dalla piattaforma, con l'obiettivo di migliorare l'utilizzo delle risorse e ridurre i costi del cloud computing.

Al momento non esiste una definizione ufficiale di serverless computing. Tuttavia, le definizioni comunemente accettate, come quelle proposte dal Berkeley View, lo descrivono come segue:

- *Serverless Computing = FaaS (Function-as-a-Service) + BaaS (Backend-as-a-Service).* Esiste un malinteso comune secondo cui il termine *serverless* può essere usato in modo intercambiabile con *FaaS*. In realtà, entrambi sono elementi fondamentali per il serverless computing. Il modello FaaS permette l'isolamento e l'invocazione delle singole funzioni, mentre il modello BaaS fornisce il supporto backend necessario per i servizi online.
- Nel modello *FaaS*, noto anche come paradigma *Lambda*, un'applicazione viene scomposta in funzioni o microservizi a livello di funzione. Gli aspetti principali che caratterizzano una funzione includono l'identificatore della funzione, il runtime del linguaggio, il limite di memoria per ciascuna istanza e l'URI (Uniform Resource Identifier) che definisce il codice della funzione.
- *BaaS* rappresenta un insieme di servizi essenziali su cui si basano le applicazioni. Alcuni esempi includono lo storage, i servizi di notifica dei messaggi e gli strumenti per il DevOps.

In sintesi, serverless computing combina sia il modello FaaS che quello BaaS, fornendo una struttura versatile per lo sviluppo e l'esecuzione di applicazioni senza la necessità di gestire direttamente l'infrastruttura sottostante.

Le funzioni cloud rappresentano quindi il fondamento del serverless computing e stanno promuovendo un modello di programmazione più semplice e versatile per il cloud. Grazie alla loro capacità di essere eseguite in risposta a eventi e richieste, le funzioni cloud permettono agli sviluppatori di concentrarsi sulla logica applicativa senza preoccuparsi della gestione dell'infrastruttura sottostante. Questo approccio semplificato consente di sviluppare, distribuire e scalare applicazioni in modo più

efficiente, aprendo la strada a un paradigma di programmazione cloud più flessibile e accessibile.

Per comprendere al meglio il modello di elaborazione serverless, si consideri un esempio basato su un'invocazione asincrona di una funzione serverless. In questo scenario, il sistema serverless riceve le API innescate e le invocazioni vengono gestite attraverso un sistema di notifica. Il sistema serverless processa le richieste API inviate dagli utenti, le valida e avvia le funzioni, creando nuove sandbox per le invocazioni (chiamato *cold startup*) oppure riutilizzando sandbox già attive (chiamato *warm startup*). Ogni invocazione di funzione viene eseguita in isolamento, all'interno di un container individuale o di una macchina virtuale, che viene assegnata da un controllore di accesso per garantire la sicurezza e l'isolamento tra le invocazioni.

Grazie alla natura *event-driven* e all'elaborazione basata su singoli eventi, il sistema serverless può essere attivato su richiesta, creando istanze isolate in risposta agli eventi e scalando orizzontalmente in base al carico effettivo dell'applicazione. Successivamente, ogni worker che esegue le funzioni accede a un database di backend per memorizzare i risultati dell'elaborazione. Gli utenti possono inoltre personalizzare l'esecuzione di applicazioni complesse configurando trigger aggiuntivi e definendo interazioni tra eventi, costruendo pipeline di eventi interni per gestire flussi di lavoro articolati.

2.3 Serverless contro approccio tradizionale

In una piattaforma serverless, l'utente si limita a scrivere una funzione cloud in un *linguaggio di alto livello* e a specificare l'evento che ne deve innescare l'esecuzione, ad esempio il caricamento di un'immagine nello storage cloud o l'inserimento di una miniatura in una tabella del database. Il sistema serverless si occupa poi di gestire tutto il resto, inclusi la selezione dell'istanza, la scalabilità, la distribuzione, la tolleranza ai guasti, il monitoraggio, la registrazione e l'applicazione di patch di sicurezza.

L'approccio tradizionale, che può essere definito anche *serverfull computing*, può essere visto come la programmazione in un linguaggio assembly di basso livello, mentre il serverless computing assomiglia alla programmazione in un linguaggio di alto livello, come Python. Un programmatore che utilizza un linguaggio assembly per calcolare un'espressione semplice come $c = a + b$ deve scegliere i registri, caricare i valori nei registri, eseguire l'operazione aritmetica e infine memorizzare il risultato. Questo processo riflette molte delle fasi del serverful computing nel cloud: prima si allocano o identificano le risorse, poi si caricano con il codice e i dati necessari, si eseguono i calcoli, si memorizzano i risultati e infine si rilasciano le risorse. L'o-

biettivo del serverless computing è quello di semplificare questo processo, offrendo ai programmatori cloud vantaggi simili a quelli della programmazione in linguaggi di alto livello.

Altre caratteristiche degli ambienti di programmazione avanzati trovano un parallelo naturale nel serverless computing. Ad esempio, la gestione automatica della memoria solleva i programmatori dal gestire le risorse di memoria; allo stesso modo, il serverless computing libera i programmatori dal dover gestire direttamente le risorse del server.

In particolare, ci sono tre differenze fondamentali tra il serverless computing e quello serverful:

- **Calcolo e storage disaccoppiati**. Nel serverless computing, lo storage e la computazione scalano in modo indipendente e vengono forniti e tariffati separatamente. In genere, lo storage è gestito tramite un servizio cloud dedicato, mentre la computazione avviene in modo stateless.
- **Esecuzione del codice senza gestione delle risorse**. L'utente non deve più preoccuparsi di allocare risorse. Basta fornire il codice, e il cloud si occupa automaticamente di assegnare le risorse necessarie per l'esecuzione.
- **Pagare per le risorse effettivamente utilizzate**. La fatturazione avviene in base alle risorse effettivamente consumate, come il tempo di esecuzione, anziché alle risorse preallocate, come la dimensione e il numero di macchine virtuali.

Usando queste differenze, si può spiegare come l'approccio serverless si distingue da soluzioni simili, presenti e passate.

L'attuale serverless computing con funzioni cloud si differenzia dai suoi predecessori per diversi aspetti essenziali: migliore autoscaling, forte isolamento, flessibilità della piattaforma e supporto dell'ecosistema dei servizi. Tra questi fattori, l'autoscaling offerto da *AWS Lambda* ha segnato un netto distacco da quanto fatto in precedenza. Ha seguito il carico con una fedeltà molto maggiore rispetto alle tecniche di autoscaling basate sui server, rispondendo rapidamente per scalare quando necessario e scendendo fino a zero risorse e zero costi in assenza di domanda. La tariffazione è molto più precisa, con un incremento minimo di fatturazione di *100 ms* in un periodo in cui altri servizi di autoscaling prevedono la tariffazione oraria.

In particolare, il cliente viene addebitato per il tempo in cui il codice *viene effettivamente eseguito*, non per le risorse riservate all'esecuzione del programma. Questa distinzione ha garantito che il cloud provider fosse "coinvolto nel gioco" dell'auto-

scaling e di conseguenza ha fornito incentivi per garantire un'allocazione efficiente delle risorse.

Per i cloud provider, il serverless computing favorisce la crescita del business, in quanto rendere il cloud più facile da programmare aiuta ad attirare nuovi clienti e a far sì che i clienti esistenti utilizzino maggiormente le offerte cloud.

Il breve tempo di esecuzione, il ridotto utilizzo della memoria e la natura stateless migliorano il multiplexing statistico, facilitando ai fornitori di servizi cloud la ricerca di risorse inutilizzate per eseguire questi compiti. I fornitori di cloud possono anche sfruttare hardware meno recente, come server più vecchi che potrebbero risultare meno interessanti per i clienti dei servizi di cloud tradizionali. Questi vantaggi contribuiscono ad aumentare il reddito derivante dalle risorse esistenti.

I clienti, dal canto loro, traggono vantaggio da una maggiore produttività nella programmazione e, in molti casi, possono anche ottenere risparmi sui costi grazie al miglior utilizzo delle risorse sottostanti. Anche se il serverless computing consente una maggiore efficienza l'uso del cloud potrebbe aumentare piuttosto che diminuire, poiché l'efficienza superiore stimola una maggiore domanda e l'ingresso di nuovi utenti.

Inoltre, il serverless computing eleva il livello di astrazione del cloud dal codice macchina *x86* (che rappresenta il 99% dei computer cloud) ai linguaggi di programmazione di alto livello, permettendo così innovazioni architettoniche. Se architetture come *ARM* o *RISC-V* offrono migliori prestazioni in termini di costi rispetto a *x86*, il serverless computing facilita il passaggio a nuovi set di istruzioni. I fornitori di cloud potrebbero persino adottare ricerche su ottimizzazioni basate sui linguaggi e su architetture specifiche per domini, al fine di accelerare l'esecuzione di programmi scritti in linguaggi come Python.

Capitolo 3

Introduzione ad AWS Lambda e Google Cloud Functions

3.1 AWS Lambda

3.1.1 Panoramica di AWS Lambda

Breve storia e introduzione di AWS Lambda.

3.1.2 Caratteristiche Principali

Descrizione delle caratteristiche principali di AWS Lambda (e.g., trigger, runtime supportati, integrazioni).

3.1.3 Deploy su AWS Lambda

Descrizione del processo di deploy di funzioni serverless su AWS.

3.2 Google Cloud Functions

3.2.1 Panoramica di Google Cloud Functions

Breve introduzione a Google Cloud Functions.

3.2.2 Caratteristiche Principali

Descrizione delle caratteristiche principali di Google Cloud Functions.

3.2.3 Deploy su Google Cloud Functions

Spiegazione del processo di deploy su Google Cloud.

Capitolo 4

Integrazione con Database NoSQL

4.1 Introduzione ai Database NoSQL

4.1.1 Caratteristiche dei Database NoSQL

Panoramica sui database NoSQL, con un focus su scalabilità, flessibilità del modello di dati e performance.

4.1.2 Vantaggi dell'Utilizzo di NoSQL in un Contesto Serverless

Spiegazione di come i database NoSQL siano particolarmente adatti per architetture serverless.

4.2 Amazon DynamoDB

4.2.1 Panoramica su DynamoDB

Introduzione a DynamoDB, il database NoSQL di AWS.

4.2.2 Integrazione di AWS Lambda con DynamoDB

Spiegazione di come le funzioni AWS Lambda interagiscono con DynamoDB, incluso l'utilizzo di trigger, accessi e operazioni CRUD (Create, Read, Update, Delete).

4.3 Google Cloud Firestore

4.3.1 Panoramica su Firestore

Introduzione a Google Cloud Firestore, il database NoSQL di Google Cloud.

4.3.2 Integrazione di Google Cloud Functions con Firestore

Spiegazione di come le funzioni Google Cloud interagiscono con Firestore, includendo l'accesso, le operazioni CRUD, e l'utilizzo di trigger.

Capitolo 5

Architettura delle API Serverless

5.1 Approccio 1: Funzione Unica per API

5.1.1 Descrizione dell'Approccio

Descrizione dell'Approccio

5.1.2 Implementazione su AWS Lambda

Implementazione su AWS Lambda

5.1.3 Implementazione su Google Cloud Functions

Implementazione su Google Cloud Functions

5.1.4 Vantaggi e Svantaggi

Vantaggi e Svantaggi

5.2 Approccio 2: Funzione per Ogni Chiamata API

5.2.1 Descrizione dell'Approccio

Descrizione dell'Approccio

5.2.2 Implementazione su AWS Lambda

Implementazione su AWS Lambda

5.2.3 Implementazione su Google Cloud Functions

Implementazione su Google Cloud Functions

5.2.4 Vantaggi e Svantaggi

Vantaggi e Svantaggi

Capitolo 6

Analisi Comparativa tra AWS Lambda e Google Cloud Functions

6.1 Performance

Tempo di Esecuzione e Latency altro?

6.2 Costi

costi, non credo ci sia bisogno di distizione tra i due approcci, il numero di chiamate dovrebbe essere lo stesso

6.3 Integrazioni e Compatibilità

magari anche facilità di collegamento tra i diversi servizi (function e db)

Capitolo 7

Caso Studio: Confronto tra le Due Soluzioni

7.1 Descrizione delle Soluzioni Software

descrizione

7.2 Implementazione su AWS Lambda

7.2.1 Deploy dell'Approccio Funzione Unica

Deploy dell'Approccio Funzione Unica

7.2.2 Deploy dell'Approccio Funzione per Ogni Chiamata

Deploy dell'Approccio Funzione per Ogni Chiamata

7.2.3 Risultati e Analisi

Risultati e Analisi

7.3 Implementazione su Google Cloud Functions

7.3.1 Deploy dell'Approccio Funzione Unica

Deploy dell'Approccio Funzione Unica

7.3.2 Deploy dell'Approccio Funzione per Ogni Chiamata

Deploy dell'Approccio Funzione per Ogni Chiamata

7.3.3 Risultati e Analisi

Risultati e Analisi

7.4 Confronto dei Risultati

7.4.1 Performance e Scalabilità

Performance e Scalabilità

7.4.2 Costi e Efficienza

Costi e Efficienza

7.4.3 Usabilità e Facilità di Deploy

Usabilità e Facilità di Deploy

Capitolo 8

Discussione dei Risultati

considerazioni generali, limiti dello studio o altro

Capitolo 9

Conclusioni

sintesi dei risultati e conclusioni

Riferimenti bibliografici