Salve a tutti, sono De Rosa Davide ed oggi vi parlerò del mio elaborato. ???

Negli ultimi anni, il Cloud Computing ha subito un’enorme evoluzione, e una delle sue innovazioni più importanti è l’introduzione del modello Serverless. Questa tecnologia promette di rendere la gestione dell’infrastruttura più semplice ed economica, permettendo agli sviluppatori di concentrarsi esclusivamente sulla logica applicativa senza preoccuparsi delle risorse fisiche o virtuali sottostanti.

I principali fornitori di servizi Cloud, come Amazon Web Services (AWS) e Google Cloud (GCP), hanno sviluppato soluzioni serverless ampiamente utilizzate nel settore accademico ed industriale, portando ad una rapida diffusione di questo modello.

L’obiettivo principale di questo elaborato è quello di condurre un’analisi comparativa tra due delle principali piattaforme per l’esecuzione di funzioni serverless: AWS Lambda e Google Cloud Functions.

Partendo dalle basi, il Serverless Computing è una tecnologia in rapida crescita che sta avendo un impatto sempre più significativo sulla società. 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.

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

* **Auto-scaling**. Ovvero la capacità di adattarsi automaticamente alle variazioni del carico di lavoro, scalando sia orizzontalmente che verticalmente. 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.
* **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. Tramite l’associazione di trigger e regole agli eventi, il sistema è in grado di rispondere in modo efficiente e flessibile alle diverse tipologie di input.
* **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.
* **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 permette di pagare solo per le risorse effettivamente utilizzate.

Un modello di calcolo che soddisfa queste cinque caratteristiche è considerato serverless.

Al momento non esiste una vera e propria definizione ufficiale di serverless computing. Tuttavia, le definizioni comunemente accettate lo descrivono come:

* **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.
* **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.

DECIDERE SE METTERE ESEMPIO DI API

Le grandi aziende tecnologiche come Amazon, Google e Microsoft offrono piattaforme serverless sotto diversi marchi. Sebbene i dettagli dei servizi possano variare, il principio fondamentale è lo stesso: con il modello di calcolo a consumo, il serverless computing mira a garantire l’autoscaling e a offrire servizi di calcolo a costi contenuti.

Sono state introdotte diverse implementazioni commerciali di successo. Amazon ha lanciato Lambda nel 2014, seguita da Google Cloud Functions, Microsoft Azure Functions e IBM OpenWhisk nel 2016.

Tutte questi servizi seguono il paradigma della programmazione funzionale: una funzione rappresenta un’unità di software che può essere distribuita sull’infrastruttura cloud del provider ed eseguire un’unica operazione in risposta a un evento esterno. Le funzioni in questione possono essere attivate da diversi tipi di eventi, come:

* un evento generato dall’infrastruttura cloud, ad esempio una modifica in un database cloud
* una richiesta diretta da parte dell’applicazione tramite HTTP o chiamate API del cloud

LAMBDA

Uno dei primi servizi di serverless computing è **AWS Lambda**, che permette di eseguire funzioni stateless scritte in uno dei linguaggi di programmazione supportati (come Node.js, Java, C# e Python) in risposta a eventi, su larga scala, con la possibilità di gestire fino a 3000 invocazioni in parallelo.

Diversamente dai tradizionali servizi Cloud IaaS, AWS Lambda elimina la necessità per gli utenti di gestire direttamente i server, offrendo un’elasticità automatica gestita dalla piattaforma. Le funzioni Lambda sono progettate per essere stateless, ovvero non dipendono dall’infrastruttura sottostante.

GOOGLE CLOUD FUNCTIONS

Pensata principalmente per i servizi di Google Cloud, Google evidenzia diversi casi d’uso specifici per Google Cloud Functions, come backend per applicazioni mobili, sviluppo di API e microservizi, elaborazione dati, webhook (per rispondere a trigger di terze parti) e applicazioni IoT. Come per le Lambda Function, viene eliminata la necessità per gli utenti di gestire direttamente i server, delegando la gestione dell’infrastruttura sottostante alla piattaforma.

CAPIRE COME GESTIRE LA SEZIONE SUI DB, FORSE RIASSUNTO DELLE DUE

ARCHITETTURA API

Nel mondo delle funzioni serverless esistono molti pattern architetturali utili per la scrittura di codice pulito, efficiente e sicuro. Due approcci utilizzati per la scrittura di funzioni serverless possono essere definiti come:

* Funzione unica per tutte le API
* Funzione per ogni chiamata API

FUNZIONE UNICA

Con il seguente approccio viene creata una singola funzione, nella quale è presente tutto il codice per gestire le diverse chiamate API presenti.

I vantaggi di questo approccio sono:

* Tutta la logica relativa al contesto viene raggruppata in un unico luogo, rendendo il codice più leggibile
* Il codice può essere riutilizzato tra le diverse funzioni
* Il security footprint è ridotto, aggiornando un singolo file permette l’aggiornamento di molte funzioni

Gli svantaggi invece sono:

* Difficoltà nel capire quando creare una nuova funzione. Ogni byte di codice extra rallenta il tempo di cold start della funzione
* Aumento del raggio d’azione delle modifiche sul codice. La modifica di una singola riga di codice potrebbe far crollare una sezione dell’infrastruttura piuttosto di una singola funzione

FUNZIONE PER OGNI CHIAMATA

Questo approccio rappresenta la forma più pura dei pattern serverless. Ogni funzione ha un suo file, ed esegue una singola chiamata API.

I vantaggi di questo approccio sono:

* Massima riusabilità del codice. Ogni funzione ha una singola responsabilità
* Si viene spinti a scrivere codice maggiormente testabile
* Minor carico cognitivo per gli sviluppatori che modificano la specifica funzione
* Miglior ottimizzazione dei tempi di esecuzione, e di conseguenza dei costi

Gli svantaggi invece sono:

* Approccio funzionante solo per architetture completamente event-driven.
* Soffermandosi sul quadro generale, il carico cognitivo aumenta quando si parla di modifiche a livello di sistema.
* La manutenzione aumenta man mano che le funzioni crescono di numero

CASO STUDIO

Prima di discutere dei risultati ottenuti dalle prove eseguite sulle piattaforme bisogna introdurre la soluzione software utilizzata e descrivere il processo di testing che ci porta ad ottenere tali risultati.

L’obiettivo della soluzione software proposta è quella di fornire funzionalità CRUD ipotizzando la presenza di un inventario da riempire con dei prodotti. Tutti i dati inerenti all’inventario verranno quindi salvati in un database.

La soluzione presenta i seguenti metodi HTTP:

* GET /items: permette di ottenere la lista di tutti i prodotti presenti nell’inventario
* PUT /items: permette di inserire o modificare un prodotto presente nell’inventario
* GET /items/{id}: permette di ottenere un singolo prodotto dell’inventario specificando un identificativo univoco, se presente
* DELETE /items/{id}: permette di eliminare un singolo prodotto dell’inventario specificando un identificativo univoco.

SEZIONE AWS LAMBDA

CHIEDERE AL PROF COME STRUTTURARE IL TUTTO