## SLIDE 1

Buongiorno a tutti, sono De Rosa Davide ed oggi vi parlerò del mio lavoro di tesi incentrata sull'analisi comparativa di soluzioni serverless.

## SLIDE 2

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.

## SLIDE 3

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

## SLIDE 4

L'obiettivo principale dell'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.

## SLIDE 5

Oltre all'esecuzione di funzioni serverless, si parlerà anche di come la persistenza dei dati si è voluta anch'essa in un modello serverless, parlando delle piattaforme Dynamo DB e Firestore.

## SLIDE 6

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.

## SLIDE 7-8-9-10-11

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.

## SLIDE 12

Le grandi aziende tecnologiche come Amazon, Google e Microsoft offrono piattaforme per l'esecuzione di funzioni 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.

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, andando a caricare il codice della funzione sulla piattaforma cloud scelta.
Successivamente, si andranno a definire i trigger della funzione, che possono essere di diverso tipo, come una chiamata API o un qualsiasi evento in un database.
Una volta configurato il tutto, la funzione potrà essere eseguita dal provider cloud quando attivata dai trigger scelti.

## SLIDE 13-14-15-16

Le limitazioni nell'utilizzo dei database nelle applicazioni basate su serverless computing richiedono un approccio del tutto innovativo alla progettazione dei database. Con il passaggio dell'architettura dal modello monolitico ai microservizi, e ora a un insieme di funzioni autonome, anche i database devono seguire gli stessi principi.
Due delle soluzioni più popolari al momento sono DynamoDB e Firestore.
Alcune delle caratteristiche principali dei database serverless possono essere:

- **Servizio completamente gestito**. La piattaforma si occupa di tutto. Recupero da guasti, crittografia dei dati, aggiornamenti, backup e altre operazioni di gestione
- **Scalabilità illimitata**. Non ci sono limiti predefiniti sulla quantità di dati che una tabella può contenere. Il pagamento avviene solo in base al consumo effettivo.
- **Flessibilità d'uso.** Non viene imposto uno schema fisso per le tabelle, permettendo agli sviluppatori di creare modelli di dati personalizzati.

## SLIDE 17

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

## SLIDE 18-19-20

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

## SLIDE 21-22-23

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

## SLIDE 24

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.

Il caso studio viene accompagnato da diversi script Python, leggermente diversi tra le due piattaforme e tra i due approcci.

Il processo di testing delle due piattaforme si basa su tre fattori principali: Performance, Costi e Usabilità-Facilità di Deploy.

## SLIDE 25-26

Per le performance si è deciso di testare la latenza dell'API.

Nel nostro caso andremo a testare la latenza ottenuta in caso di cold startup e non, osservando le possibili differenze anche tra i due approcci di API.

La fase di testing consiste nel misurare la latenza effettuando una media di 10 chiamate, escludendo i due valori estremi.

In entrambi i casi viene testato il metodo GET /items.
I risultati ottenuti per Lambda sono:
foto tabella lambda

I risultati ottenuti per Cloud Functions sono:
foto tabella cloud functions

## SLIDE 27-28

Per i costi si è deciso di osservare il costo per singola richiesta alla funzione. I costi vengono dichiarati dai due provider, dove entrambi offrono un numero di richieste gratuite al mese. Lambda ha un costo di 0,23$ per 1 milione di richieste, mentre Cloud Functions ha un costo di 0,40$ per 1 milione di richieste.
Lambda include inoltre 1 milione di richieste gratuite al mese, mentre Cloud Functions ne offre 2 milioni.

## SLIDE 29

Come ultimo fattore di confronto si è deciso di parlare dell'usabilità della piattaforma e della facilita di deploy.
Per quanto riguarda l'usabilità, entrambe le piattaforme presentano interfacce intuitive, curate e dettagliate. AWS fornisce di gran lunga una miglior documentazione per lo sviluppatore, fornendo esempi di piccoli progetti. GCP risulta più carente di documentazione ufficiale.

Per il discorso facilita di deploy, le Cloud Functions richiedono un minor numero di step per avere una funzione pronta all'uso, portando ad un setup iniziale più rapido rispetto a Lambda. Quest'ultima presenta una prima configurazione iniziale più complessa e dettagliata, portando ad un processo di deploy più lungo rispetto a Cloud Functions.

SLIDE 30

Per quanto riguarda le performance, le Lambda Functions tendono ad avere una latenza minore, sia in casi di cold startup che non, rispetto alle Cloud Functions.

È possibile osservare inoltre come i due approcci di scrittura delle API non portino quasi alcun tipo di differenza sulla latenza nel nostro caso studio, potendo quindi affermare che in casi con funzioni non molto lunghe è indifferente l'approccio scelto. Ovviamente, all'aumentare della lunghezza della funzione, la funzione unica arriverà ad avere latenze maggiori.

Durante la fase di testing è stato possibile notare anche un altro piccolo dettaglio. Lambda esegue cold startup molto più frequentemente di Cloud Functions, dove la funzione tende a reagire in maniera più reattiva anche dopo un periodo di inattività più lungo.

Per quanto riguarda i costi, AWS offre di gran lunga un costo più vantaggioso, inferiore a GCP di 0,17$ per milione di richieste.

Va però ribadito il numero di richieste gratuite al mese offerto da entrambe le piattaforme, dove Google offre il doppio delle richieste rispetto ad Amazon. In progetti di dimensioni ridotte, o con un numero di richieste mensili ridotto, la possibilità di non pagare alcuna cifra al di sotto delle 2 milioni di richieste potrebbe portare piccoli team di sviluppo a scegliere la piattaforma offerta da Google.

Per concludere, la facilita di deploy offerta da entrambe le piattaforme è ppiù che soddisfacente, con Cloud Functions che risulta essere più immediata da configurare e subito pronta all'uso.
Un aspetto dove Amazon vince con un netto distacco è la presenza di documentazione ufficiale e di progetti d'esempio ben fatti, i quali consentono agli sviluppatori alle prime armi nel mondo serverless di avere esempi di qualità come riferimento.

Per concludere, dovendo proclamare una sorta di "vincitore" tra le due piattaforme, mi sento di affermare che AWS Lambda sia leggermente migliore di GCP Cloud Functions, con costi per richiesta inferiori e miglior documentazione, nonostante una più lunga fase di deploy.

Questa conclusione è in linea con l'utilizzo generale delle piattaforme nel mondo serverless, dove AWS ha una percentuale di utilizzo nettamente maggiore rispetto ai competitors, coprendo circa l'80% del mercato.