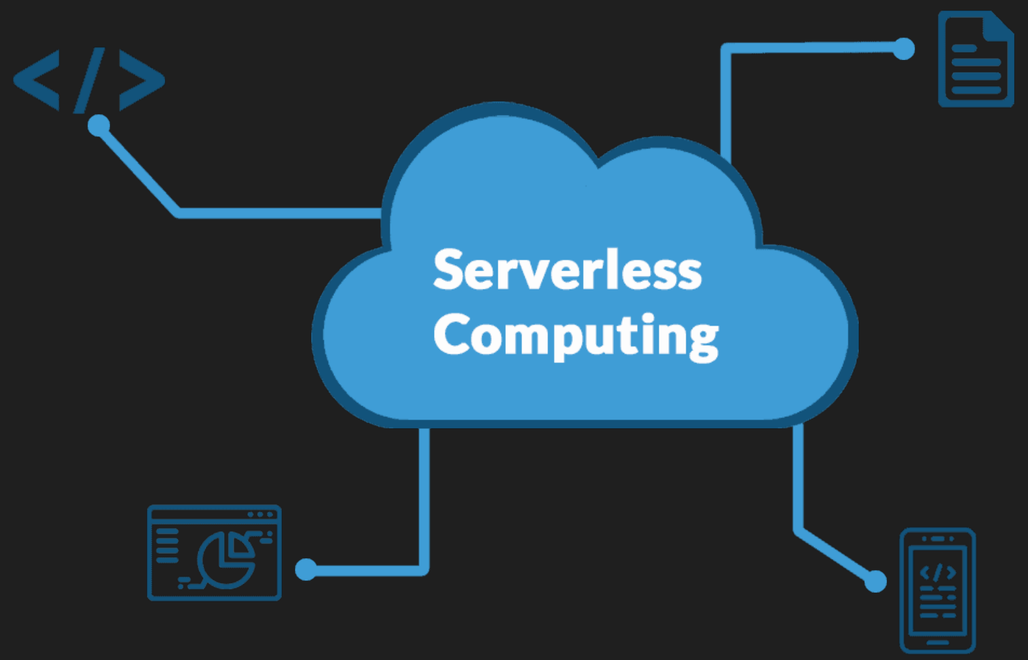
Università di Bologna

Analisi comparativa di soluzioni Serverless

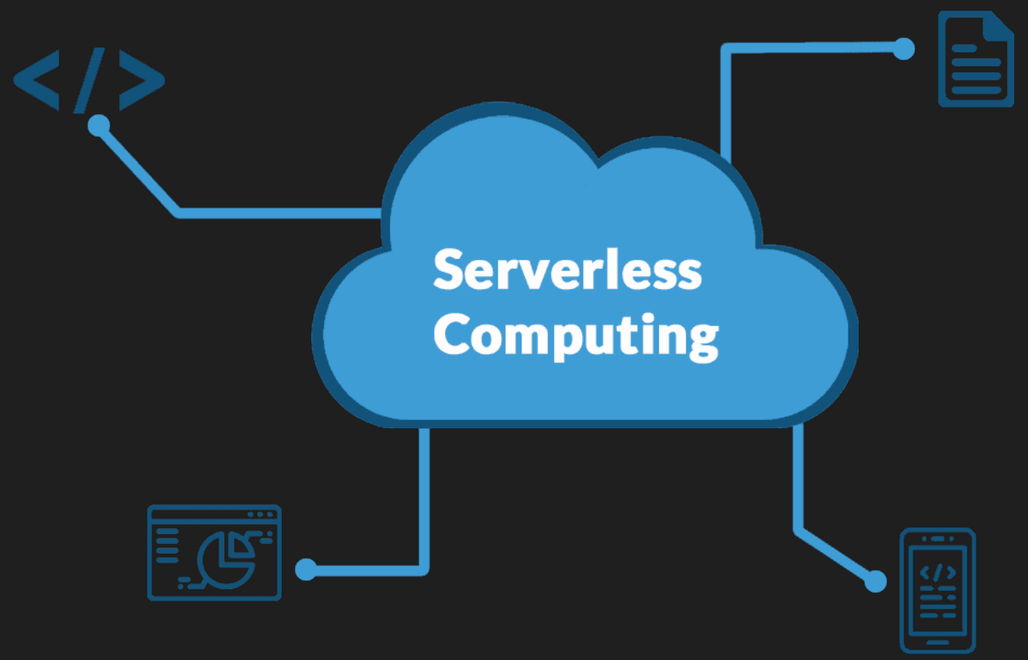
Relatore:
Davide Rossi

Presentata da:
Davide De Rosa

Serverless Computing



Serverless Computing



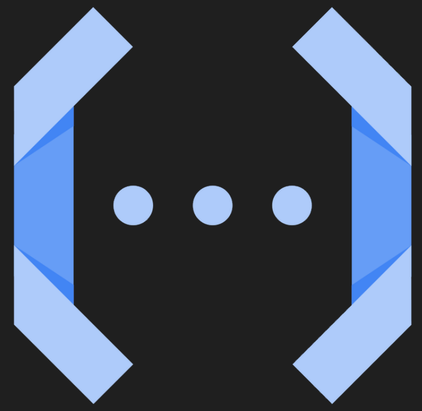
aws

Google Cloud

Analisi di soluzioni Serverless

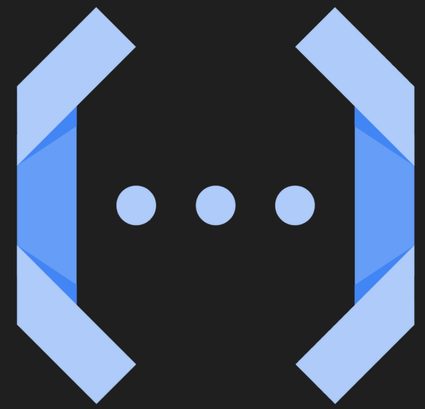
AWS Lambda

Google Cloud Functions



AWS Lambda

Google Cloud Functions



AWS DynamoDB

Google Firestore

Il Serverless Computing è una tecnologia in rapida crescita. La sua promessa principale è quella di:

- Rendere i servizi informatici più accessibili
- Personalizzabili in base alle esigenze specifiche
- Delegare all’infrastruttura la gestione operativa

Auto-scaling

Adattamento alle variazioni di carico possibili, scalando sia orizzontalmente che verticalmente.
Quando il numero di istanze arriva a zero, si parla di *cold startup*.

Caratteristiche di un Sistema Serverless

Auto-scaling

Adattamento alle variazioni di carico possibili, scalando sia orizzontalmente che verticalmente.
Quando il numero di istanze arriva a zero, si parla di *cold startup*.

Pianificazione flessibile

Nessun vincolo ad un server specifico, garantendo bilanciamento del carico e prestazioni ottimali.

Caratteristiche di un Sistema Serverless

Auto-scaling

Adattamento alle variazioni di carico possibili, scalando sia orizzontalmente che verticalmente.
Quando il numero di istanze arriva a zero, si parla di *cold startup*.

Pianificazione flessibile

Nessun vincolo ad un server specifico, garantendo bilanciamento del carico e prestazioni ottimali.

Event-Driven

Il sistema si attiva in risposta ad eventi, come richieste HTTP, aggiornamenti di code di messaggi o scritture su servizi di storage.

Caratteristiche di un Sistema Serverless

Auto-scaling

Adattamento alle variazioni di carico possibili, scalando sia orizzontalmente che verticalmente.
Quando il numero di istanze arriva a zero, si parla di *cold startup*.

Pianificazione flessibile

Nessun vincolo ad un server specifico, garantendo bilanciamento del carico e prestazioni ottimali.

Event-Driven

Il sistema si attiva in risposta ad eventi, come richieste HTTP, aggiornamenti di code di messaggi o scritture su servizi di storage.

Sviluppo trasparente

Responsabilità della gestione delle risorse fisiche e dell'ambiente di esecuzione delegata al provider.

Caratteristiche di un Sistema Serverless

Auto-scaling

Adattamento alle variazioni di carico possibili, scalando sia orizzontalmente che verticalmente.
Quando il numero di istanze arriva a zero, si parla di *cold startup*.

Pianificazione flessibile

Nessun vincolo ad un server specifico, garantendo bilanciamento del carico e prestazioni ottimali.

Event-Driven

Il sistema si attiva in risposta ad eventi, come richieste HTTP, aggiornamenti di code di messaggi o scritture su servizi di storage.

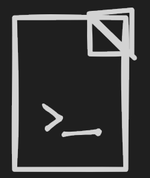
Sviluppo trasparente

Responsabilità della gestione delle risorse fisiche e dell'ambiente di esecuzione delegata al provider.

Pagamento a consumo

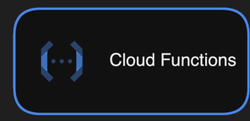
Il modello Serverless permette di pagare solo per le risorse effettivamente utilizzate, eliminando la necessità di acquistare server dedicati.

Funzioni Serverless



Carica il codice della funzione sulla piattaforma Cloud

Definisci i trigger della funzione (come una chiamata API o un evento in un database)



La funzione viene eseguita dal provider Cloud quando attivata

Database Serverless

AWS DynamoDB

Google Firestore

Database Serverless

AWS DynamoDB

Google Firestore

Servizio gestito

La piattaforma si occupa di tutto. Recupero da guasti, crittografia dei dati, aggiornamenti, backup e altre operazioni di gestione.

Database Serverless

AWS DynamoDB

Google Firestore

Servizio 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.

Database Serverless

AWS DynamoDB

Google Firestore

Servizio 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.

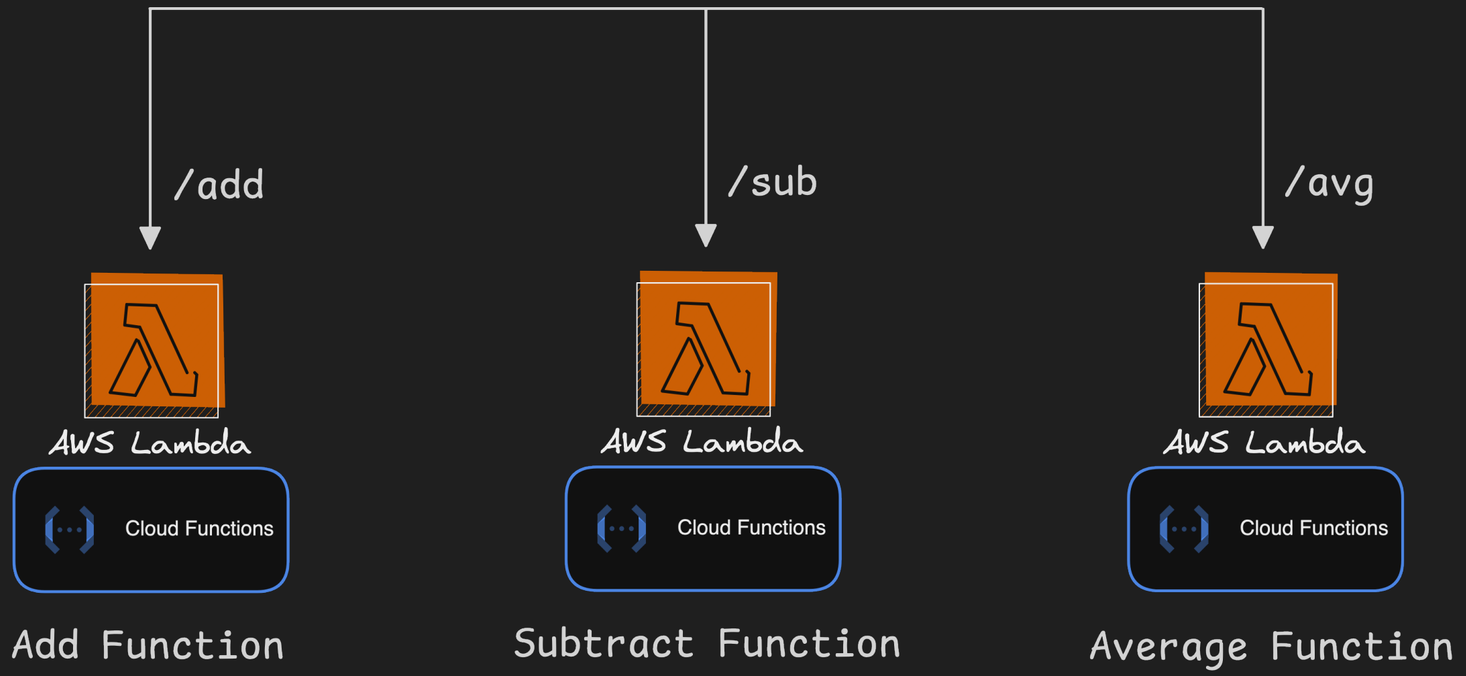
Flessibilità d’uso

Non viene imposto uno schema fisso per le tabelle, permettendo agli sviluppatori di creare modelli di dati personalizzati.

Tra le possibili architetture per lo sviluppo di API, sono state confrontate due possibili soluzioni:

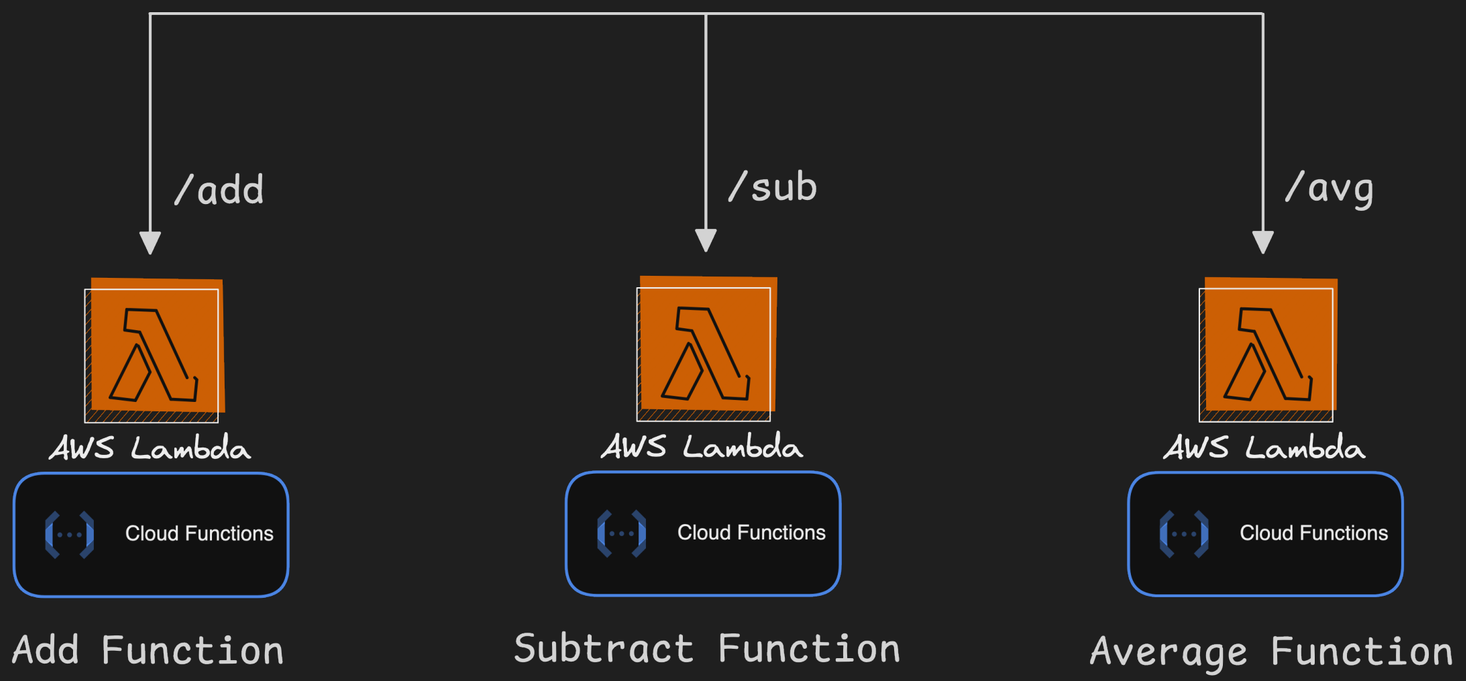
- Funzione unica per tutte le API
- Funzione per ogni chiamata API

Funzione unica



funzione_unica.py

Funzione unica



funzione_unica.py

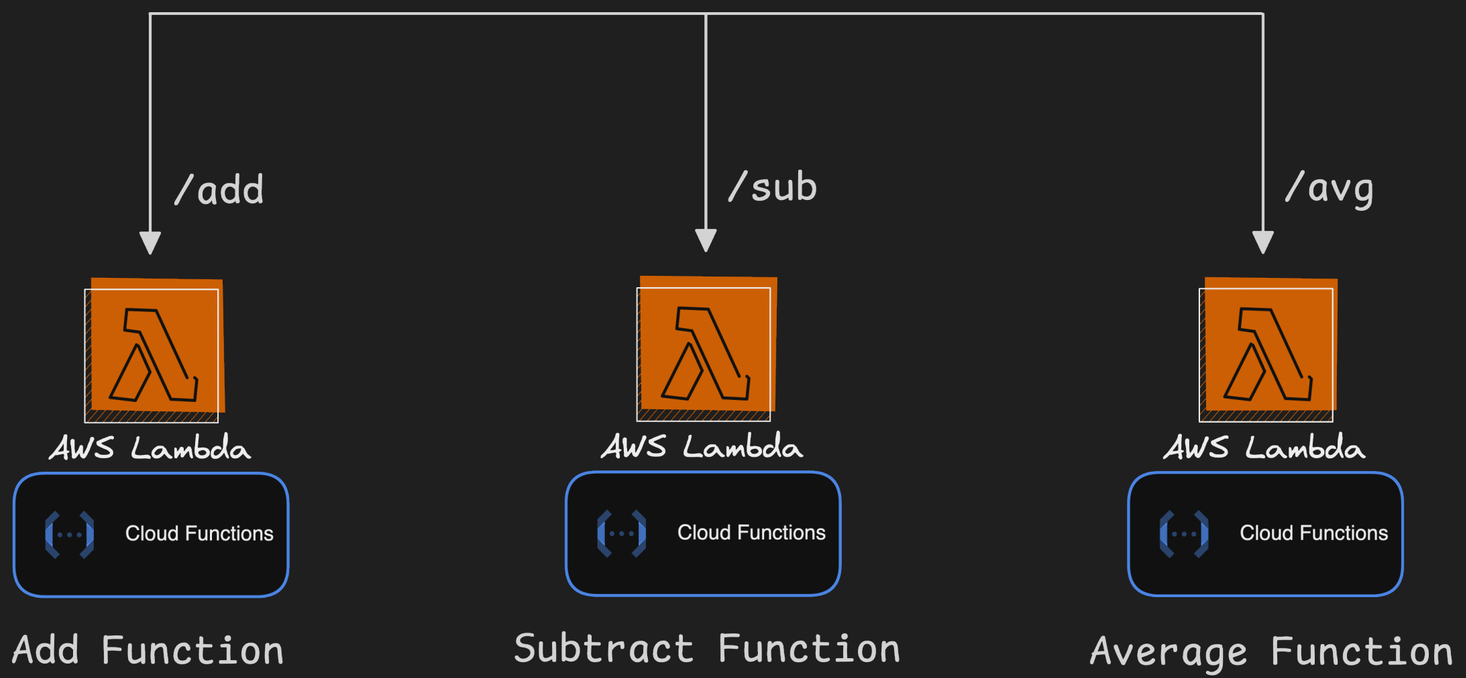
Vantaggi

Logica raggruppata in un unico luogo

Codice più leggibile e riutilizzabile

Security footprint ridotto, aggiornando un singolo file si aggiornano molte funzioni

Funzione unica



funzione_unica.py

Vantaggi

Logica raggruppata in un unico luogo

Codice più leggibile e riutilizzabile

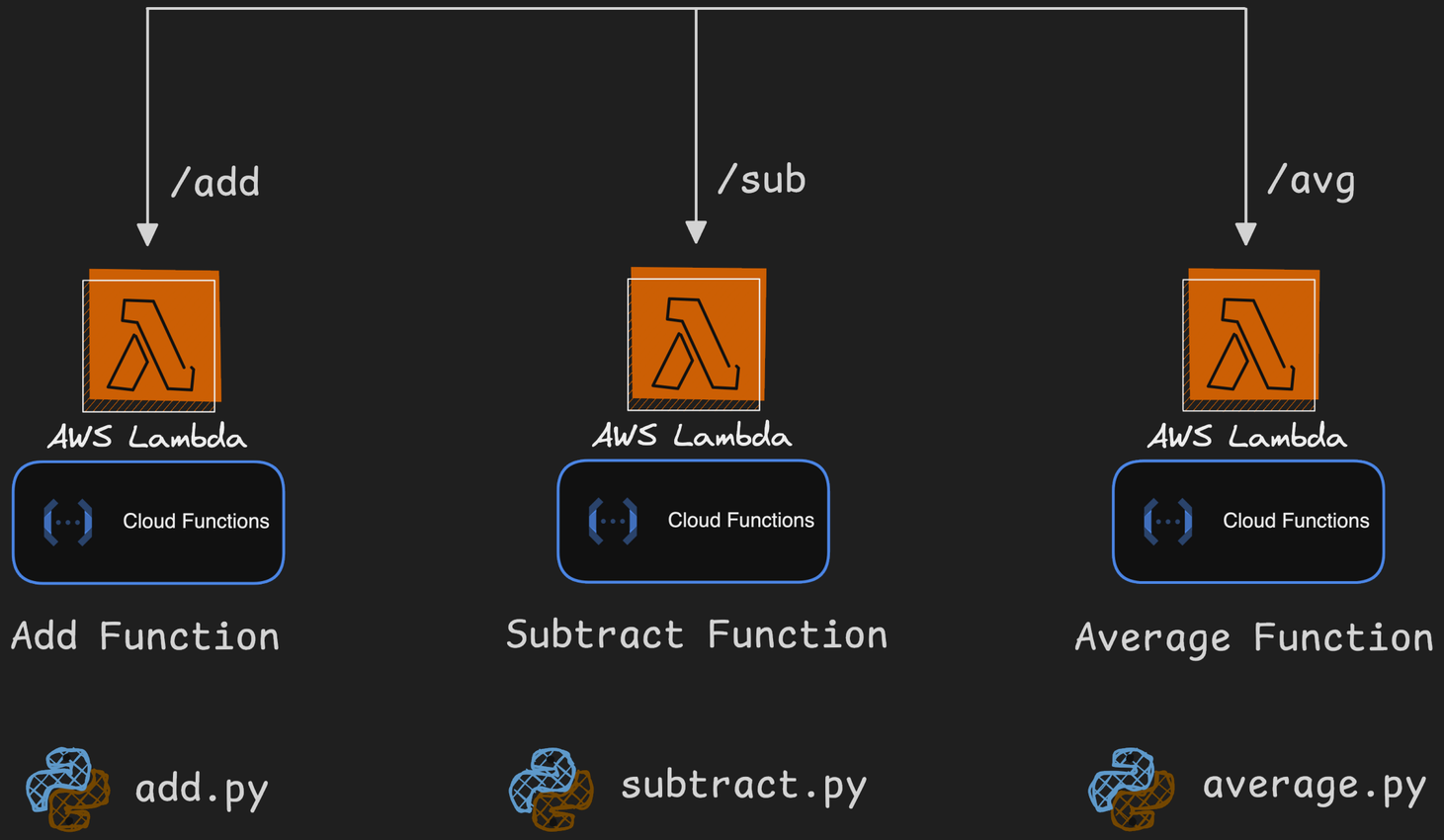
Security footprint ridotto, aggiornando un singolo file si aggiornano molte funzioni

Svantaggi

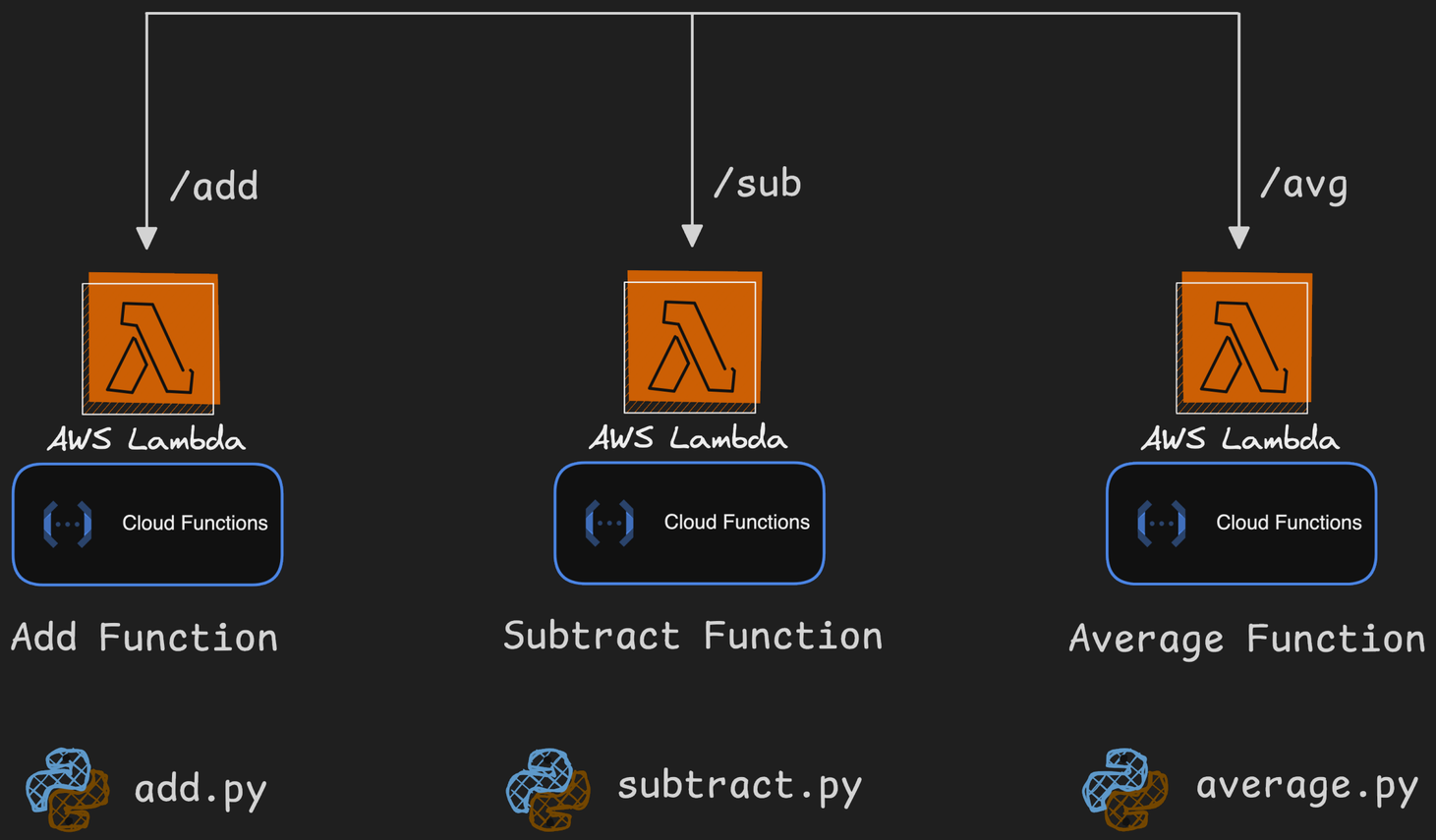
Difficoltà nel capire quando creare una nuova funzione

Aumento del raggio d’azione delle modifiche sul codice

Funzione per ogni chiamata



Funzione per ogni chiamata



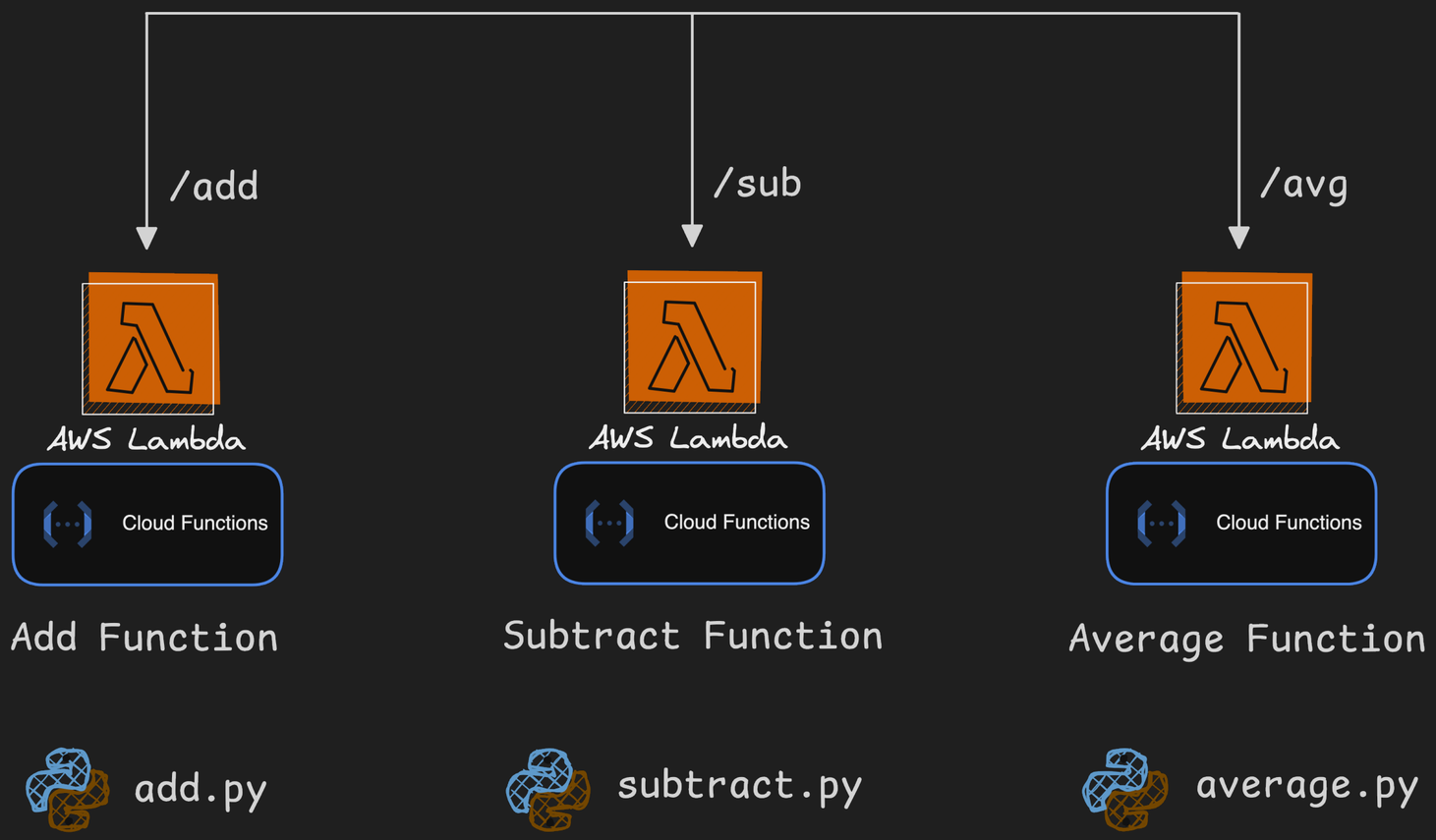
Vantaggi

Massima riusabilità del codice

Porta alla scrittura di codice più testabile, seguendo il *Single Responsability Principle*

Ottimizzazione dei tempi di esecuzione, e di conseguenza dei costi

Funzione per ogni chiamata



Vantaggi

Massima riusabilità del codice

Porta alla scrittura di codice più testabile, seguendo il *Single Responsability Principle*

Ottimizzazione dei tempi di esecuzione, e di conseguenza dei costi

Svantaggi

Approccio funzionante solo per architetture completamente *Event-Driven*

Manutenzione maggiore quando il numero di funzioni diventa elevato

Caso Studio

GET	/items	}	Read
GET	/items/{id}		
PUT	/items	}	Create, Update
DELETE	/items/{id}	}	Delete

Latenza

Il tempo che un’API impiega per elaborare una richiesta ed inviare una risposta, compresi eventuali ritardi di rete o di elaborazione. Può essere influenzata da vari fattori, come la velocità della connessione di rete, il tempo di elaborazione dell’API e la quantità di dati trasferiti.

Latenza

Il tempo che un'API impiega per elaborare una richiesta ed inviare una risposta, compresi eventuali ritardi di rete o di elaborazione. Può essere influenzata da vari fattori, come la velocità della connessione di rete, il tempo di elaborazione dell'API e la quantità di dati trasferiti.

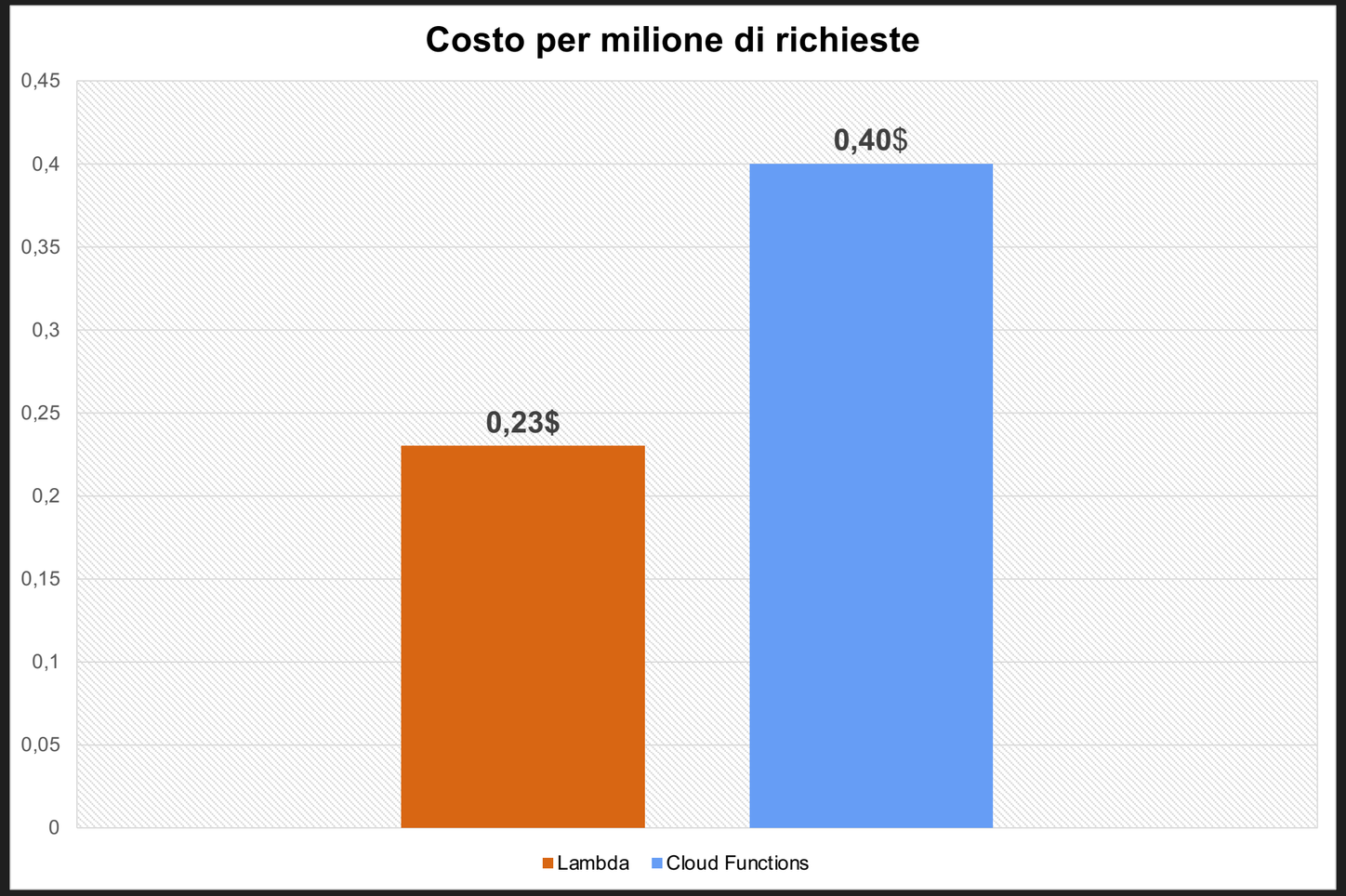
AWS Lambda

	Funzione Unica	Funzione Singola
Cold Startup	585 ms	556 ms
Normale	84 ms	86 ms

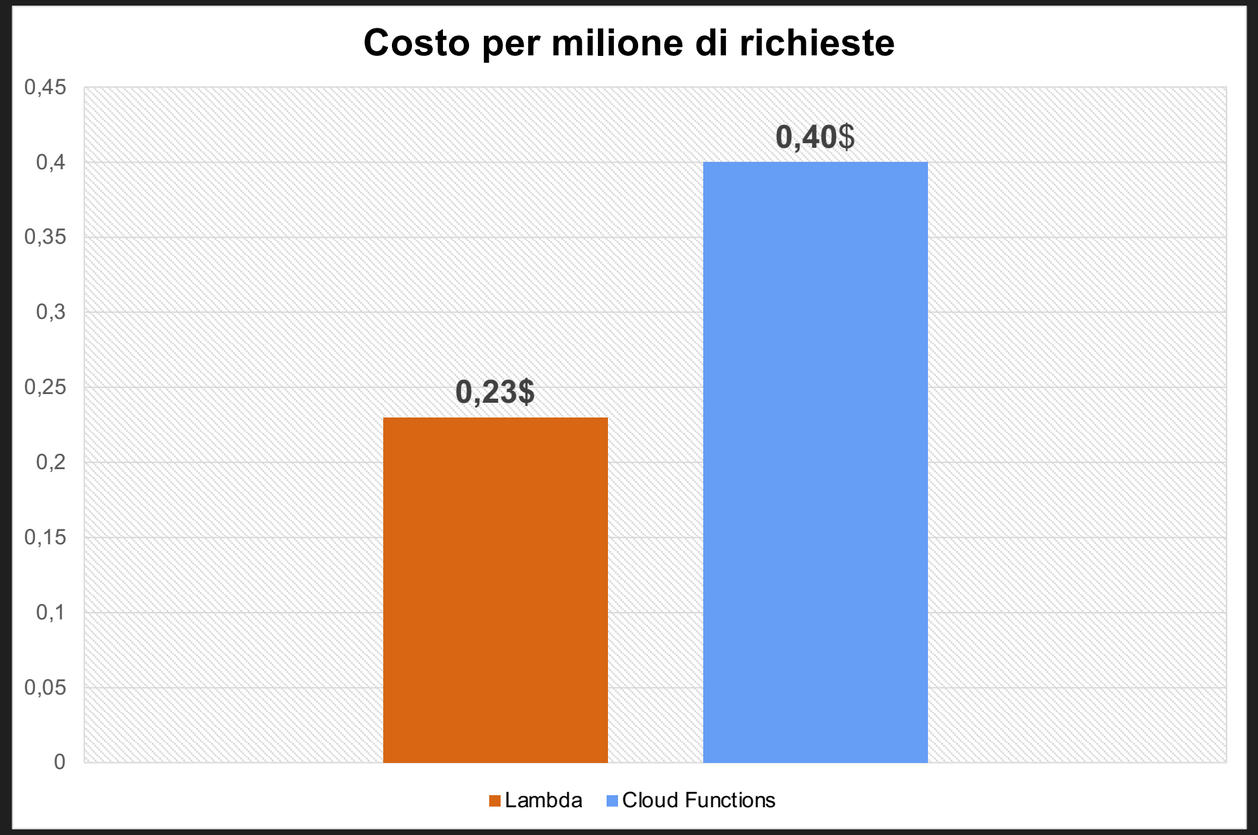
Google Cloud Functions

	Funzione Unica	Funzione Singola
Cold Startup	857 ms	802 ms
Normale	113 ms	106 ms

Caso Studio - Costi



Caso Studio - Costi



Richieste gratuite

Lambda: 1 milione di richieste al mese

Cloud Functions: 2 milioni di richieste al mese

Caso Studio - Usabilità/Facilità di Deploy

AWS Lambda

Usabilità

Interfaccia intuitiva, curata e dettagliata

Migliore documentazione per gli sviluppatori

Esempi pratici per piccoli progetti software

Facilità di Deploy

Processo di deploy più lungo rispetto a GCF

Configurazione più complessa e dettagliata della funzione per il setup iniziale

G.C. Functions

Usabilità

Interfaccia intuitiva, curata e dettagliata

Documentazione ufficiale più carente rispetto a Lambda

Facilità di Deploy

Minor numero di step per il deploy

Setup iniziale più rapido rispetto a Lambda

Caso Studio - Confronto dei risultati

	AWS Lambda	Google Cloud Functions
Performance	+	-
Costo per richiesta	+	-
Richieste gratuite	-	+
Facilità di Deploy	-	+
Qualità della documentazione	+	-

Grazie per l’attenzione!