ANALISI COMPLETA DI AWS LAMBDA: COME OTTIMIZZARE I PICCHI E PREVENIRE I COLD START

AWS Lambda | DevOps | Serverless

beSharp | 20 Marzo 2020

Quando si parla di paradigma Serverless, molti sono gli aspetti che dobbiamo tenere in considerazione per evitare problemi di latenza e produrre così applicazioni più belle, più robuste e sicure. In questo articolo discuteremo di molti aspetti da tenere in considerazione quando si decide di sviluppare in AWS Lambda, come evitare alcuni dei problemi più comuni ma anche di come sfruttare le innovazioni introdotte recentemente da Amazon per creare app Serverless più efficienti, performanti e meno costose.

Cold Start

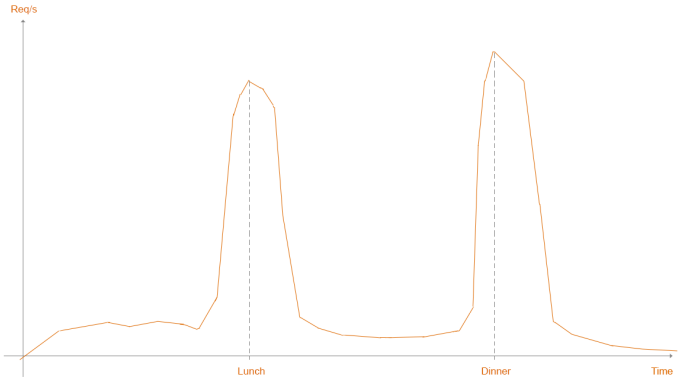
Per anni l'argomento cold start è stato uno dei più accesi e più frequentemente dibattuti della community di Serverless.

Supponiamo che abbiate provvisionato una nuova funzione Lambda. Indipendentemente dal modo in cui la funzione è invocata, una nuova Micro VM deve essere istanziata; questo perchè non ci sono ancora istanze disponibili per rispondere a questo preciso evento. L'unione dei tempi necessari a fare il setup del Runtime di Lambda, insieme con il vostro codice e le sue dipendenze, viene comunemente chiamato Cold Start.

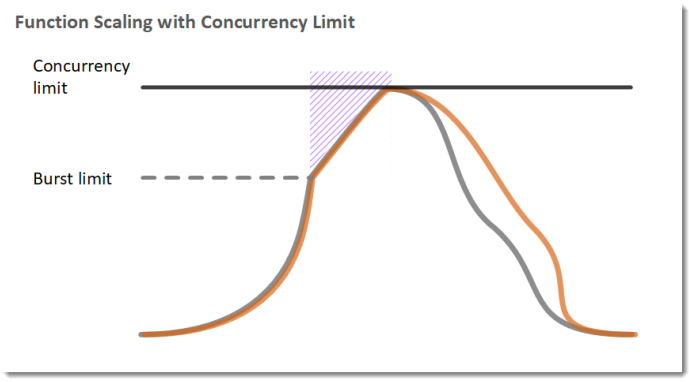
Indipendentemente dal tipo di Runtime scelto, un processo di setup può impiegare almeno dai 50 ai 200 ms prima che l'esecuzione abbia realmente luogo. Lambda scritte in Java e .Net possono anche dar vita a cold start di diversi secondi!

Facendo riferimento al vostro caso d'uso, i cold start possono diventare sicuramente un collo di bottiglia, prevenendo così la possibile adozione di un paradigma Serverless per la vostra applicazione. Per fortuna, per molti sviluppatori, questo problema è facilmente aggirabile perché il carico di lavoro della loro soluzione è prevedibile e stabile o, in alcuni casi, basato su calcoli ad uso esclusivamente interno, ad es. data-processing.

La documentazione di AWS ci mostra un ottimo esempio per comprendere meglio i cold start e i problemi correlati con le nostre necessità di scalare. Immaginiamo di avere a che fare con società come JustEat e Deliveroo solite a ricevere picchi di traffico durante gli orari di pranzo e cena.



Questi picchi causano il raggiungimento potenziale dei limiti di una applicazione come ad esempio quanto velocemente AWS Lambda sia in grado di scalare oltre il suo limite iniziale di **burst-capacity**. Dopo il **burst** iniziale, una lambda può scalare linearmente fino a 500 istanze per minuto per servire richieste concorrenti. Prima di poter servire tali richieste, però, ogni nuova istanza si trova ad affrontare un cold start. Limiti di concurrency e una alta latenza di risposta dovuta ai cold start possono far sì che le vostre politiche di scaling non siano in grado di far fronte al traffico, causando l'eventuale scarto di nuove richieste.

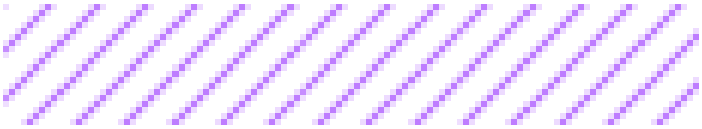


courtesy of AWS

Legenda

Istanze per le funzioni

Richieste aperte



possibile Throttling

Reserved Concurrency

La Concurrency di AWS Lambda ha un limite per regione, condiviso da tutte le funzioni in quella specifica regione, altro punto da tenere in considerazione quando diverse Lambda sono soggette a invocazioni molto frequenti. Facendo riferimento alla tabella sottostante abbiamo:

Limiti di Burst Concurrency

- **3000** – US Ovest (Oregon), US Est (N. Virginia), Europa (Irlanda)
- **1000** – Asia Pacifico (Tokyo), Europa (Francoforte)
- **500** – Altre regioni

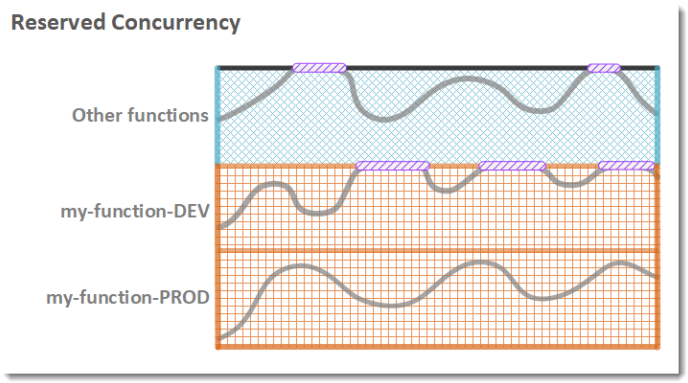
Per garantire che una specifica funzione possa sempre raggiungere uno specifico livello di concurrency e per restringere il numero di istanze di una Lambda function aventi accesso a risorse di basso livello (ad es. Un database), è possibile configurare la **reserved concurrency**.

Quando una funzione ha abilitato la **reserved concurrency**, quest'ultima ha sempre la possibilità di aumentare il numero di istanze al valore specificato nella configurazione della reserved concurrency, indipendentemente dall'utilizzo delle altre Lambda. La Reserved Concurrency si applica ad una funzione Lambda nella sua interezza, alias e versioni comprese.

Per riservare la concurrency per una funzione, basta seguire questi semplici passi:

1. Aprire la console di "**Lambda Functions"** e selezionare una funzione.
2. Alla voce "**Concurrency"**, selezionare Reserve concurrency.
3. Digitare il valore di concurrency da riservare per la funzione e salvare.

L'esempio successivo mostra come la reserved concurrency possa aiutare a mitigare il throttling delle richieste.

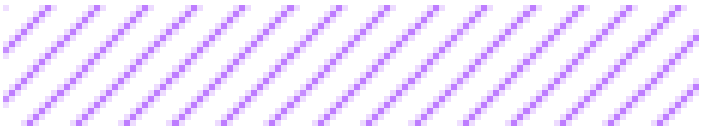


courtesy of AWS

Legenda

Istanze per le funzioni

Richieste aperte



possibile Throttling

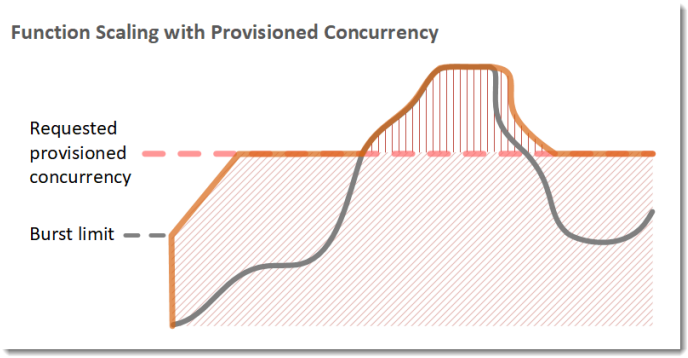
La Reserved Concurrency può essere applicata quando si ha una comprensione chiara del numero massimo possibile di richieste e del tasso di concorrenza, ma non risolve comunque i problemi dovuti ai cold start poiché ogni nuova istanza creata per sostenere la concorrenzialità delle richieste incapperà in quel problema.

Prima che AWS rilasciasse la feature di Provisioned Concurrency, cercare di evitare o almeno mitigare il problema del cold start per rendere le Lambda più responsive era un compito difficile: bisognava far uso di codice custom lato utente per capire o stato di una Lambda se pronta on in stato di warming. Più avanti alcune librerie sono salite alla ribalta come lambda-warmer e serverless-plugin-warmup, tuttavia non possono essere considerate una soluzione pulita e definitiva.

Provisioned Concurrency

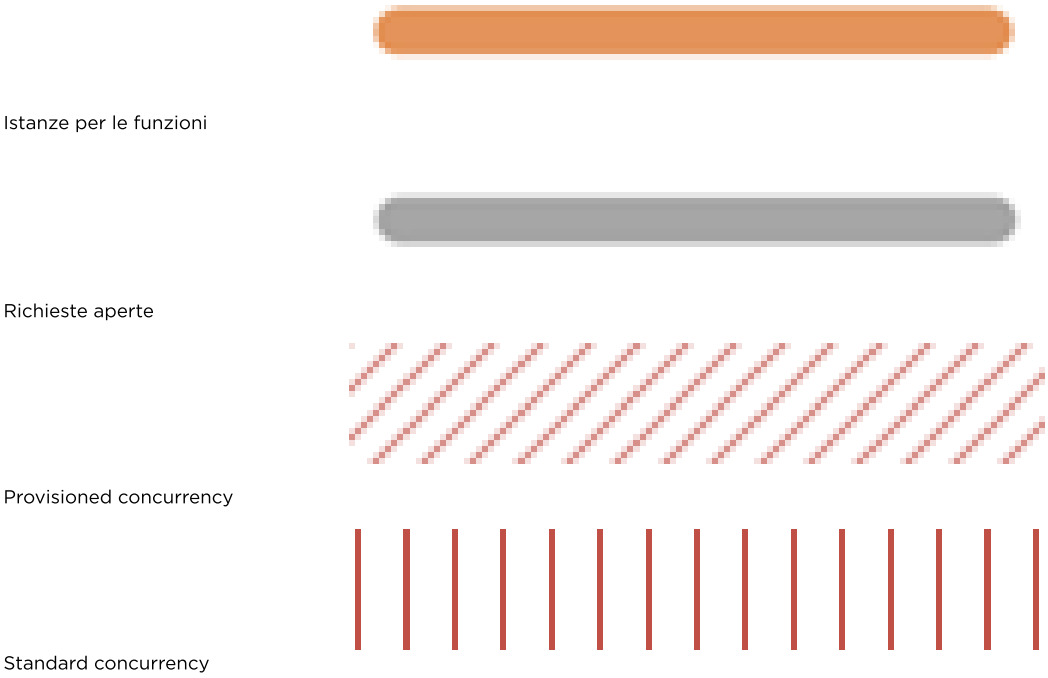
Per rendere capace una funzione di scalare senza fluttuazioni in latenza, si utilizza la **provisioned concurrency**. Quest'ultima opzione permette di configurare un certo numero di istanze già "calde" fin dall'inizio e che non richiede interventi sul vostro codice applicativo.

Il seguente esempio mostra una funzione con abilitata la provisioned concurrency elaborare un singolo picco di traffico.



courtesy of AWS

Legenda



Quando la **provisioned concurrency** è allocata, la funzione è in grado di scalare con la stessa politica di burst della concurrency di default.

Una volta allocata, la **provisioned concurrency**, gestisce le richieste in arrivo con una latenza molto bassa. Quando tutta la provisioned concurrency è in uso, la funzione prenderà a scalare normalmente per gestire le ulteriori richieste. Queste richieste addizionali incorreranno nel cold start, ma saranno un numero significativamente basso se la Provisioned Concurrency è stata configurata ad hoc.

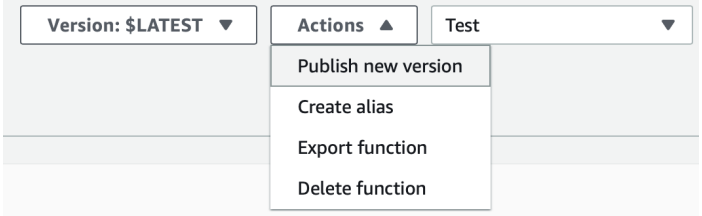
Grazie a questa nuova feature è ora possibile migrare ad un paradigma Serverless per servizi storicamente difficili da migrare, come:

- APIs che girano su Runtime Java/.Net
- Microservizi con molte comunicazioni API-to-API.
- API con un andamento di traffico ricco di picchi non prevedibili.

Si può configurare la Provisioned Concurrency sia per gli Alias che per una Versione specifica. Se viene configurata per un Alias, tutte le versioni ad esso associate erediteranno il valore di Provisioned Concurrency. Altrimenti ogni versione avrà la propria configurazione indipendente. In questo modo è possibile associare differenti versioni di Lambda , con differenti valori di Provisioned Concurrency, a diversi carichi di traffico, non solo, è possibile sfruttare questa peculiarità per effettuare AB testing.

E' bene notare che non è possibile configurare la Provisioned Concurrency per l'Alias $LATEST o qualsiasi Alias che a lui faccia riferimento.

Per pubblicare una nuova versione di Lambda, è sufficiente entrare nella pagina di dettaglio della Lambda desiderata e cliccare "Publish new version" dalle azioni del menù a tendina in alto allo schermo, come mostrato in figura:

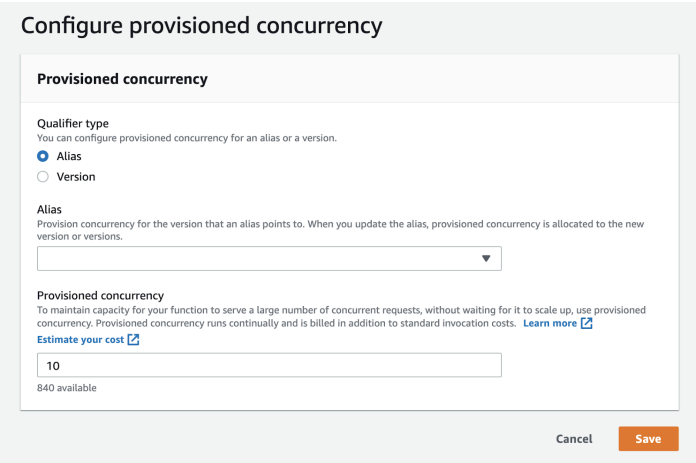


Dopo una pubblicazione, la nuova versione è settata:

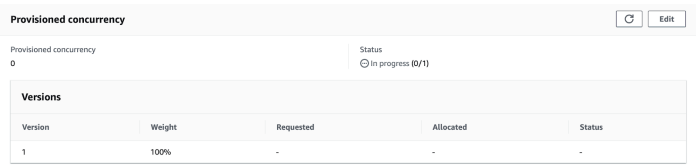
A questo punto è possibile configurare la Provisioned Concurrency per la versione appena creata (Version 1) ma, per maggior esaustività, esploreremo il metodo per configurare la Provisioned Concurrency per un nuovo Alias che punta a “Version 1”.

Procedendo, sotto la sezione **Aliases**, cliccare su “**+ Create alias**”. Questo comando apre un nuovo modale nel quale è possibile creare un nuovo alias, che punterà alla versione appena creata.

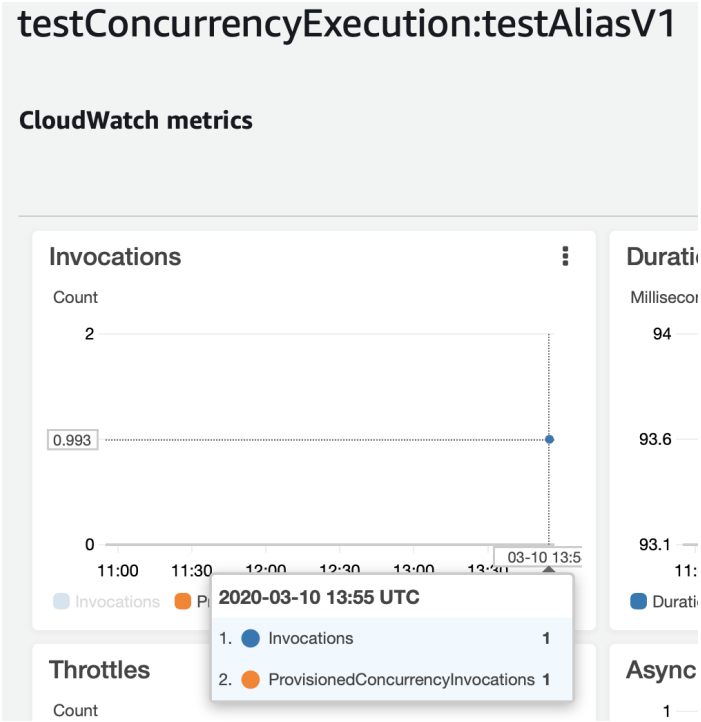
Quando l’alias è stato creato, è possibile infine configurare la Provisioned Concurrency.



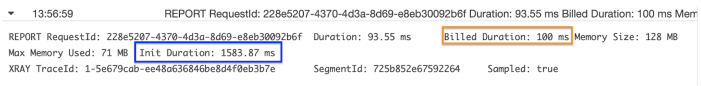
Puntiamo ora alla sezione “Provisioned Concurrency”. Qui possiamo configurare la **Provisioned Concurrency** per il nuovo alias, inserendo un valore che rappresenta quante esecuzioni parallele possiamo provisionare dal nostro Pool di Reservation. E’ davvero molto semplice, basta solo avere l’accortezza di sapere che questo ovviamente corrisponde ad un incremento dei costi così come specificato da AWS.



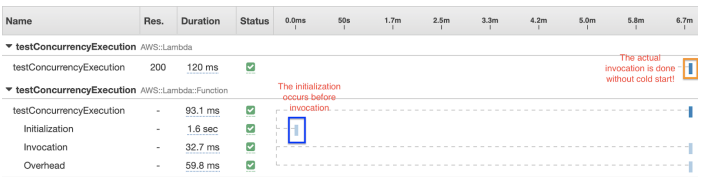
Una volta definita la Provisioned Concurrency, il valore della colonna “Status” nella sezione apposita, cambierà in **Ready**. Le invocazioni della Lambda a questo punto saranno gestite dalla Provisioned Concurrency a monte di quella on-demand Standard.



Nel grafico, possiamo vedere che le invocazioni di Lamdba sono effettivamente ora gestite da istanze provisionate e se analizziamo anche CloudWatch Logs, possiamo vedere chiaramente come tale invocazione di Lambda riporti la dicitura **Init Duration**, che corrisponde al tempo necessario per permettere a Lambda di effettuare il setup del numero richiesto di istanze per le esecuzioni. Oltre a questo abbiamo la voce classica di **Billed Duration**.



Possiamo notare la stessa cosa anche prendendo in esame la console di X-Ray per questa invocazione di Lambda.



Volendo compiere un ulteriore miglioramento possiamo attivare anche l'opzione di **autoscaling per la provisioned concurrency**. Quando usiamo l'Application Auto Scaling, possiamo creare una **target tracking scaling policy** che modifica il numero di esecuzioni contemporanee basandosi su metriche di utilizzo fornite da Lambda.

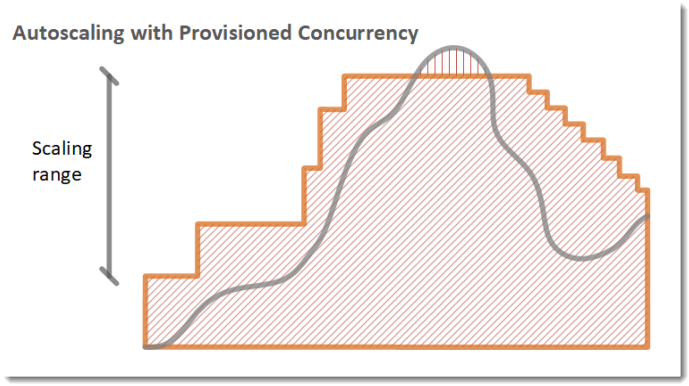
Nota a margine. Sembra non esserci un modo semplice per cancellare una configurazione di Provisioned Concurrency dalla console di AWS, per tale motivo si può usare questo comando della cli di AWS:

```
aws lambda delete-provisioned-concurrency-config --function-name
<function-name> --qualifier <version-number/alias-name>
```

Auto Scaling API

E’ possibile utilizzare l’API di Application Auto Scaling per registrare un alias come un **scalable target** a cui associare una **policy di scaling**.

Nell’esempio seguente, una funzione scala tra un valore minimo e massimo di **provisioned concurrency** basandosi sull’utilizzo della funzione stessa; al crescere del numero di richieste aperte , l’Application Auto Scaling aumenta la Provisioned Concurrency a grossi step fino a raggiungere il massimo configurato.

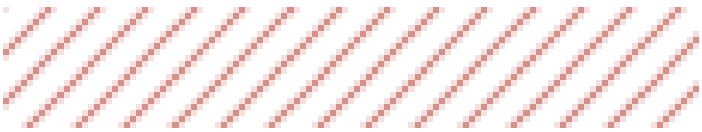


courtesy of AWS

Legenda

Istanze per le funzioni

Richieste aperte



Provisioned concurrency

Standard concurrency

La funzione prosegue scalando con la standard concurrency finchè l’utilizzo della stessa non comincia a calare. Quando l’utilizzo è di nuovo sufficientemente e consistentemente basso, l’Application Auto Scaling diminuisce la Provisioned Concurrency in piccoli gradini periodici (metà destra dell’immagine).

A parte l’auto scaling basato su metriche di utilizzo AWs consente anche di schedulare le politiche di Scaling. In entrambi i casi, è prima necessario registrare un alias come scaling target.

Metriche di Lambda

Nuove e migliorate metriche di AWS Lambda sono ora disponibili per meglio aiutare a definire i picchi di traffico così come le Lambda possano rispondere in termini di concorrenza e invocazioni simultanee.

Metriche di Concurrency

- `ConcurrentExecutions`

  – Il numero di stanze per le funzioni che stanno processando gli eventi. Se il numero raggiunge la vostra concurrent executions limit per una regione specificata o il reserved concurrency limit che è stato configurato per quella funzione, le richieste successive saranno scartate.

- `ProvisionedConcurrentExecutions`

  – Il numero di stanze per le funzioni che stanno processando eventi sotto provisioned concurrency. Per ogni invocazione sotto l'alias o la versione con provisioned concurrency, Lambda aumenta il contatore corrente.

- `ProvisionedConcurrencyUtilization`

  – Per una versione o un alias, il valore delle ProvisionedConcurrentExecutions divise per il valore totale della provisioned concurrency allocata. Ad esempio,

  ```
  .5
  ```

  indica che il 50 percento della provisioned concurrency allocata è in uso.

- `UnreservedConcurrentExecutions`

  – Per una regione di AWS, il numero di eventi che stanno venendo processati da funzioni che non hanno la reserved concurrency.

Fino ad ora, abbiamo parlato su come attivare la Provisioned Concurrency in risposta alle nostre necessità di Lambda veloci e scalabili, ma come DevOps, probabilmente vorremmo avere la possibilità di manipolare questi parametri come parti di codice. Di seguito due esempi per Serverless e AWS SDK.

Esempio per Serverless

In Serverless si può attivare la **provisionedConcurrency** e la **reservedConcurrency** come descritto nella documentazione, di seguito un semplice esempio con tutti i parametri che si possono

settare:

```
service: myService
provider:
  name: aws
  runtime: nodejs12.x
  memorySize: 512 # optional, in MB, default is 1024
  timeout: 10 # optional, in seconds, default is 6
  versionFunctions: false # optional, default is true
  tracing:
    lambda: true # optional, enables tracing for all functions (can be true (true equa
ls 'Active') 'Active' or 'PassThrough')
functions:
  hello:
    handler: handler.hello # required, handler set in AWS Lambda
    name: ${self:provider.stage}-lambdaName # optional, Deployed Lambda name
    description: Description of what the lambda function does # optional, Description
 to publish to AWS
    runtime: python2.7 # optional overwrite, default is provider runtime
    memorySize: 512 # optional, in MB, default is 1024
    timeout: 10 # optional, in seconds, default is 6
    provisionedConcurrency: 3 # optional, Count of provisioned lambda instances
    reservedConcurrency: 5 # optional, reserved concurrency limit for this function. B
y default, AWS uses account concurrency limit
    tracing: PassThrough # optional, overwrite, can be 'Active' or 'PassThrough'
```

Nell'esempio appena mostrato, la funzione Lambda hello avrà sempre 3 **istanze in stato warm,** subito pronte a gestire le chiamate HTTP in arrivo da API Gateway.

Tuttavia si può fare anche di più! Si può scrivere una Lambda che predisposta a partire ogni ora con uno scheduling ben preciso in mente. Infatti se site come molte società, avrete sicuramente chiaro gli orari dei picchi e non vorrete la provisioned concurrency sempre attiva, ma solo durante quei picchi di traffico prevedibili.

La Provisioned Concurrency può anche essere settata nell' **AWS SDK**:

```
const AWS = require('aws-sdk');
const params = {
FunctionName: 'Hello',
ProvisionedConcurrentExecutions: '3',
Qualifier: 'your-alias-here'
};
const lambda = new AWS
lambda.putProvisionedConcurrencyConfig(params, function(err, data) {
if (err) { console.log(err, err.stack); } // an error occurred
else { console.log(data); } // successful response
});
```

Ora avete gli strumenti per schedulare la provisioned concurrency nel modo che preferite e di conseguenza anche ottimizzare i suoi costi, vediamo come!

Saving Plan per AWS Lambda

Per invogliare ulteriormente potenziali utenti all'adozione della Provisioned Concurrency, Compute Saving Plans ora include AWS Lambda, insieme a Amazon EC2 e AWS Fargate. Il servizio prevede sconti extra per un commitment su una forbice di tempo da 1 a 3 anni per quanto riguarda l'uso intensivo di risorse computazionali.

I vantaggi di usare Compute Saving Plans è che il cambiamento richiesto è basato solo su risorse computazionali e quindi permette tranquillamente di cambiare, taglio, tipologia e caratteristiche di tali risorse.

Utilizzando Saving Plans, le funzioanlità **Lambda usage for Duration**, **Provisioned Concurrency**, e **Duration (Provisioned Concurrency)** saranno prezzate con sconti fino al 17% come documentato da AWS.

Anche se non è previsto uno sconto specifico per le **Requests usage**, quest'ultimo viene comunque coperto da Saving Plans, permettendo dunque ai clienti di meglio gestire il loro commitment.

Conclusioni

Finalmente una delle limitazioni di AWS Lambda più fortemente dibattute è diventata cosa passata, è queste sono ottime notizie per tutti i DevOps e in generale per chiunque sia desideroso di adottare il paradigma Serverless.

Per concludere in questo post si è parlato di:

- Il problema dei cold start e come questi siano bloccanti per alcuni sviluppatori su AWS.
- Che cos'è la Provisioned Concurrency e come funziona?
- Come funziona con l' auto-scaling?
- Alcuni snippet per abilitare la Provisioned concurrency via templating
- Come Saving Plans può aiutare a ridurre i costi

Spero vi siate divertiti a leggere questo post. Contattateci per approfondire!

Arrivederci al prossimo articolo 🙂

beSharp

Dal 2011 beSharp guida le aziende italiane sul Cloud. Dalla piccola impresa alla grande multinazionale, dal manifatturiero al terziario avanzato, aiutiamo le realtà più all'avanguardia a realizzare progetti innovativi in campo IT.

Get in touch

beSharp.it
proud2becloud@besharp.it