DISACCOPPIARE SERVIZI CON SQS E LAMBDA TRIGGER

Amazon Simple Queue Service (SQS) | AWS Lambda | DevOps | Microservices | Serverless

beSharp | 24 Gennaio 2020

La progettazione di un'applicazione basata su microservizi presenta alcune importanti sfide da affrontare come implementare sistemi di service discovery, la standardizzazione della comunicazione interna tra di essi, la loro sincronizzazione e molto altro ancora.

Uno degli scenari più comuni è il disaccoppiamento di due servizi quando il primo si basa sull'altro per l'elaborazione asincrona di alcuni dati. Possiamo risolvere questo tipo di problemi utilizzando un classico approccio di tipo producer-consumer.

Nel Cloud, possiamo sfruttare i managed services per facilitare la realizzazione di tale architettura. L'infrastruttura di riferimento per queste situazioni consiste nell'utilizzo una o più code SQS per disaccoppiare il producer dal consumer.

I managed services garantiscono numerosi benefici tra cui:

- Nessun bisogno di verificare la disponibilità del consumer;
- Metriche affidabili relative alle code;
- Nessuna necessità di implementare logiche complesse per adattare la portata del servizio producer;
- I messaggi possono essere storati fino a che il onsumer non sarà pronto per utilizzarli.

Prima di addentrarci nella realizzazione della soluzione su AWS è doveroso fare una veloce carrellata sui servizi che andremo ad utilizzare più tardi. Cominciamo!

Cos'è Amazon SQS?

Amazon Simple Queue Service (SQS) è un servizio di code completamente gestito che permette di disaccopiare e far scalare microservizi, sistemi distribuiti e applicazioni Serverless.

Essendo un servizio gestito, la sua applicazione libera totalmente il team di sviluppo dalla complessità di gestione di un middleware message-oriented e permette agli sviluppatori di concentrarsi sulle attività core.

Utilizzando le API è possibile inviare, memorizzare e ricevere messaggi tra i componenti software a qualsiasi volume, senza perdere messaggi e non dipendendo dalla disponibilità di altri servizi.

SQS offre due tipi di code di messaggi:

- Code standard: offrono la massima velocità di trasmissione, il massimo impegno nell'ordinazione e un delivery di tipo *at-least-once delivery.*
- Code FIFO: sono progettate per garantire che i messaggi vengano elaborati esattamente secondo l'ordine esatto in cui sono inviati.

Cos'è AWS Lambda?

AWS Lambda è un servizio di calcolo che consente di eseguire il codice senza dover effettuare il provisioning delle risorse e senza dover gestire alcun server.

AWS Lambda esegue il codice solo quando necessario e scala automaticamente a seconda delle richieste sostenendo efficacemente qualunque carico di lavoro, da poche richieste al giorno a migliaia al secondo.

Il servizio si occupa anche di eseguire il codice su nfrastrutture di calcolo in alta disponibilità by design e di eseguire tutta l'amministrazione delle risorse di calcolo: dalla manutenzione dei server e del sistema operativo, al provisioning della capacità, dallo scaling automatico, al monitoraggio e al logging del codice.
Da tenere presente è il fatto che non è possibile accedere alle istanze di calcolo o personalizzare il sistema operativo nei tempi di esecuzione previsti.

Ora che i servizi sono stati introdotti, passiamo ad esplorare la soluzione.

Il problema

Per rendere la discussione più pragmatica, facciamo finta di essere nella seguente situazione:

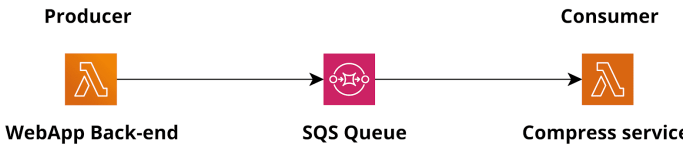
Abbiamo un'applicazione web serverless che permette agli utenti di selezionare documenti e di scaricarli. Per ottimizzare i tempi di trasferimento occorre costruire un servizio per creare un archivio compresso contenente i file e salvarlo su S3. Il servizio creerà anche un link firmato per consentire il download dell'archivio.

In questa occasione non implementeremo nessuna delle funzionalità, ma ci limiteremo ad utilizzare un po' di codice stub per dimostrare la soluzione. In ogni caso, avere una situazione realistica ci

aiuterà a capire del dettaglio il meccanismo di funzionamento e i potenziali problemi che possono sorgere.

La soluzione

Proponiamo ora la tipica infrastruttura AWS per il modello producer-consumer basata su AWS Lambda and SQS.



Affinché tutto funzioni, è necessario configurare un trigger SQS per il consumer; in questo modo il consumer lambda verrà attivato nel momento in cui saranno rilevati messaggi sulla coda.

Tornando alla nostra situazione di esempio, quindi:

Il producer lambda sarà la funzione lambda nel backend dell'applicazione web adibita all'invio di una richiesta al servizio di compressione, mentre il consumer sarà un altro microservizio basato su Lambda col compito di leggere una lista di file (forse URI s3) e di omprimerli prima di memorizzare il risultato su S3. Supponiamo inoltre che questo servizio generi e memorizzi anche un link firmato in un database, da utilizzare successivamente.

Vediamo ora il flusso di base per l'implementazione del pattern producer-consumer:

il servizio di back-end basato su Lambda utilizzerà AWS SDK per effettuare una chiamata API di tipo SendMessage verso SQS aggiungendo un messaggio alla coda. Il consumer Lambda verrà invocato automaticamente dal servizio Lambda quando il messaggio sarà stato aggiunto alla coda. A questo punto, il consumer procederà ad evadere la richiesta ricevuta "consumando" il messaggio.

Come funziona un trigger SQS?

La funzione sarà invocata dal servizio Lambda che riceverà i messaggi come parametro di input.

Per le code standard, Lambda sfrutta il long-polling standard (una chiamata chiamata ogni 20 secondi) di SQS per interrogare una coda fino a quando questà non diventa attiva.

Quando i messaggi saranno disponibili, Lambda sarà in grado di leggere fino a 5 batch e li invierà alla nostra funzione. La dimensione di ciascun batch non è altro che il numero di messaggi inviati ad una Lambda e può essere modificata nelle impostazioni di trigger (1 – 10). Modificando il paramentro mentre sono ancora disponibili dei messaggi, Lambda aumenterà il numero di processi dedicati alla lettura dei batch (fino a 60 istanze in più al minuto). Il numero massimo di batch che

possono essere elaborati contemporaneamente da una mappatura della sorgente dell'evento è 1000.

Per le code FIFO, Lambda invia messaggi alla funzione nell'ordine in cui li riceve.

Quando si invia un messaggio a una coda FIFO, si specifica un ID del gruppo di messaggi. Amazon SQS assicura che i messaggi dello stesso gruppo siano consegnati a Lambda in ordine. Lambda ordina i messaggi in gruppi e invia un solo lotto alla volta corrispondente a ciascun gruppo. Nel caso in cui la funzione restituisca un errore, saranno effettuati tutti i tentativi sui messaggi interessati prima che Lambda possa ricedvere ulteriori messaggi dallo stesso gruppo.

Quindi, fondamentalmente, se si configura il trigger con una dimensione di batch di 1, Lambda eseguirà il polling della coda e invocherà una funzione ogni volta che un nuovo lavoro sarà disponibile su di essa. Ogni lambda riceverà esattamente 1 lavoro.
Se configurate un batch di 5, Lambda invocherà comunque la funzione il prima possibile, ma fino a 5 messaggi alla volta saranno consegnati.

Quando il servizio lambda riceverà un messaggio, dovrà eseguire il lavoro e ritornare un risultato senza errori affinchè il messaggio venga correttamente eliminato dalla coda.

E se la funzione fallisse invece?

Le cause di errore possono essere le più varie, dal caso in cui la funzione Lambda non disponga di spazio su diosco sufficiente alla presenza di URI di Input non validi nel messaggio.

Cosa succede?

Semplice: se non è possibile per qualsiasi motivo completare il lavoro, il messaggio resta non disponibile per il tempo di timeout (configurabile e personalizzabile). Al termine del periodo di timeout, poi, il messaggio tornerà disponile e sarà pronto per essere elaborato da un altro consumer innescando nuovamente il timeout. Si entrerebbe così in un (potenzialmente) infinito e costosissimo loop. Per evitare di rimanere "intrappolati" in caso di errori, si potrebbe semplicemente recuperare tutte le eccezioni all'interno del codice Lambda e salvare il fallimento in modo tale che il messaggio possa comunque essere considerato "consumato".

Esiste però un modo per affrontare i fallimenti nel modo migliore possibile

Dead letter queues

In ogni coda di SQS è presente un'impostazione opzionale per specificare il parametro Dead letter queues.

Una coda di tipo dead letter è una coda speciale in cui SQS inserisce automaticamente i messaggi che vengono rifiutati per un numero di volte configurabile.

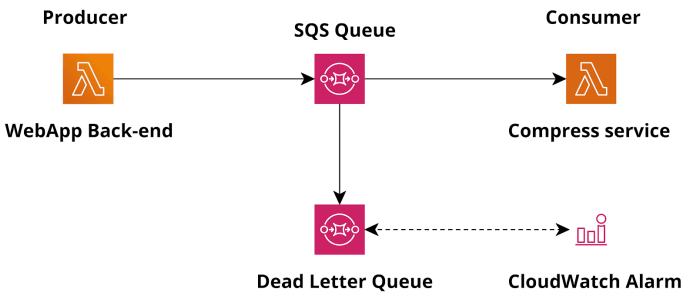
Così, ad esempio, è possibile creare una coda aggiuntiva e specificare il suo ARN come DLQ per la coda principale. È anche possibile specificare il numero massimo di tentativi – "maxReceiveCount" –

nella coda principale, in modo che il nostro sistema consenta solo alcuni tentativi. Questo è in realtà molto utile, perché, come già accennato, un messaggio può essere rifiutato per innumerevoli ragioni, alcune delle quali non sono legate al codice del consumer e sono probabilmente temporanee. Un esempio è il raggiungimento della soglia-limite di Lambda currency.

La soluzione aggiornata

Tornando alla nostra situazione: aggiungere una dead letter queue ci aiuterà a risparmiare tempo e denaro.

Un altro accorgimento potrebbe essere la configuarzione di un allarme CloudWatch che avverta il team di sviluppo quando il numero di messaggi rifiutato eccede un certo limite.



Un esempio funzionante

Ecco un semplice template CloudFormation che crea una coda, una coda dead letter e una funzione lambda configurata per essere attivata sui nuovi messaggi.

```
AWSTemplateFormatVersion : 2010-09-09
Resources:

 LambdaExecutionRole:
   Type: AWS::IAM::Role
   Properties:
     AssumeRolePolicyDocument:
       Version: '2012-10-17'
       Statement:
       - Effect: Allow
         Principal:
           Service:
           - lambda.amazonaws.com
         Action:
         - sts:AssumeRole
     Policies:
       - PolicyName: allowLambdaLogs
         PolicyDocument:
           Version: '2012-10-17'
           Statement:
           - Effect: Allow
             Action:
             - logs:*
```

```
            Resource: arn:aws:logs:*:*:*
        - PolicyName: allowSqs
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
            - Effect: Allow
              Action:
              - sqs:ReceiveMessage
              - sqs:DeleteMessage
              - sqs:GetQueueAttributes
              - sqs:ChangeMessageVisibility
              Resource: !GetAtt MyQueue.Arn

  LambdaConsumer:
    Type: AWS::Lambda::Function
    Properties:
      Code:
        ZipFile: |
          def lambda_handler(event, context):
              for record in event['Records']:
                print(record['body'])
      Handler: index.lambda_handler
      Role: !GetAtt LambdaExecutionRole.Arn
      Runtime: python3.7
      Timeout: 10
      MemorySize: 128

  LambdaFunctionEventSourceMapping:
    Type: AWS::Lambda::EventSourceMapping
    Properties:
      BatchSize: 1
      Enabled: true
      EventSourceArn: !GetAtt MyQueue.Arn
      FunctionName: !GetAtt LambdaConsumer.Arn

  MyQueue:
    Type: AWS::SQS::Queue
    Properties:
      DelaySeconds: 0
      VisibilityTimeout: 30
      RedrivePolicy:
        deadLetterTargetArn : !GetAtt DLQ.Arn
        maxReceiveCount : 3
   DLQ:
    Type: AWS::SQS::Queue
    Properties:
      DelaySeconds: 0
      VisibilityTimeout: 180
```

Ora è possibile implementare e testare il template inserendo un qualsiasi messaggio nella coda e cercando i log di esecuzione di lambda.

```
START RequestId: 45dddad8-49d3-4378-ba7a-2e03217e9c40 Version: $LATEST
Hello from SQS!
END RequestId: 45dddad8-49d3-4378-ba7a-2e03217e9c40
```

```
REPORT RequestId: 45dddad8-49d3-4378-ba7a-2e03217e9c40  Duration: 1.56 ms       Billed
Duration: 100 ms        Memory Size: 128 MB     Max Memory Used: 56 MB  Init Duration:
119.17 ms
```

In questo articolo abbiamo esplorato uno dei modelli più comuni per disaccoppiare i microservizi serverless utilizzando servizi managed ad alta disponibilità su AWS ottenendo un'architettura estremamente flessibile, resiliente e solida.

Soddisfatti? Fateci sapere! 😉

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