Migrazione graduale e refactoring di applicazioni verso l'approccio serverless attraverso l'utilizzo di Amazon API Gateway

24 Dicembre 2021 - 6 min. read

Amazon API Gateway | Cloud Migration | Serverless

Oggigiorno sempre più applicazioni, servizi e progetti vengono sviluppati seguendo l'approccio serverless.

Molteplici applicazioni potrebbero sfruttare il suddetto approccio per migliorare le proprie performance e al contempo ottenere alta disponibilità ed elasticità. Tuttavia, è necessario che queste vengano migrate e rifattorizzate nel dettaglio.

Migrare un'applicazione verso il paradigma serverless implica spesso una parziale riscrittura della codebase al fine di implementare un modello di esecuzione che sia adatto al FaaS selezionato. Inoltre, l'architettura complessiva dovrebbe accogliere l'approccio a microservizi e un flusso basato sugli eventi favorendo, dove possibile, il processamento asincrono e il disaccoppiamento attraverso servizi di messaggistica.

Lo sforzo e le tempistiche richieste per ottenere le sopra citate specifiche, specialmente nel caso di applicazioni complesse, possono facilmente raggiungere elevati ordini di grandezza. A volte, inoltre, è preferibile non rifattorizzare completamente un'applicazione prima del suo rilascio.

Il presente articolo vi illustrerà una tecnica che, attraverso API Gateway, si occuperà del routing e dell'adattamento delle richieste dello user, permettendo la graduale migrazione e la rifattorizzazione di applicazione complesse senza compromettere la loro disponibilità.

Perché API Gateway?

Amazon API Gateway è un servizio managed che rappresenta la front door della propria applicazione. Il servizio funziona come un gateway managed per il backend, può essere anche direttamente integrato con altri servizi AWS supportati per indirizzare il carico verso il processore adeguato o un sistema di messaggistica.

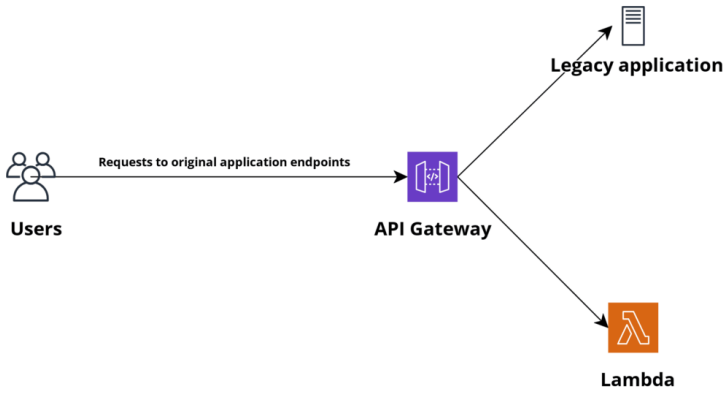
API Gateway si occupa di centinaia di migliaia di chiamate API concorrenti, gestisce il traffico, il CORS, l'autorizzazione, il controllo degli accessi, il throttling, il monitoraggio e il versionamento delle API.

La tecnica utilizzata in questo articolo sfrutta le potenzialità di API Gateway per farlo agire da proxy per l'intero applicativo, per accettare richieste, apportare le modifiche necessarie al formato del payload ed eseguire il routing tra l'applicazione legacy e gli endpoint rifattorizzati.

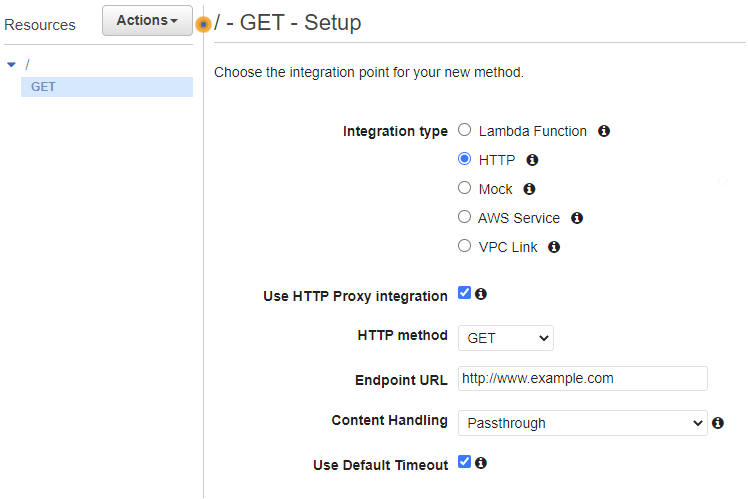
Replatforming e rifattorizzazione nella teoria

Il metodo proposto in questo articolo permette di mappare tutti gli endpoint dell'applicazione sottoposti a migrazione verso un unico API Gateway, il quale agirà da punto di accesso centralizzato dell'applicazione.

Una volta che l'intero flusso di richieste si dirige verso un singolo oggetto infrastrutturale, sarà possibile mantenere due versioni distinte dell'applicazione: il modello legacy e la nuova versione, che possiamo implementare incrementalmente.



Una volta mappati tutti gli endpoint all'API Gateway, è necessario configurarlo per proxare l'infrastruttura dell'applicazione corrente. Per farlo, possiamo usare un'integrazione specifica chiamata "HTTP", per la quale è necessario abilitare l'opzione proxy come mostrato nell'immagine seguente.



A questo punto, API Gateway può diventare l'unico punto di accesso dell'applicazione. Dovrebbe essere possibile azionare il DNS per puntare il dominio al nuovo punto d'ingresso e vedere tutto il traffico fluire attraverso il Gateway senza che sia necessaria alcuna modifica dell'applicazione.

Terminata l'appena citata configurazione, è possibile rielaborare alcune componenti dell'applicazione; una buona strategia consiste nel migrare a singoli microservizi ove possibile. In caso di servizi estesi o applicazioni monolitiche, vi è la possibilità di migrare singoli endpoint, rifattorizzarli secondo l'approccio serverless e cambiare le impostazioni di questi endpoint per instradare le richieste alla nuova versione.

È possibile operare anche un semplice A/B testing, raccogliere le metriche inerenti alle performance dei nuovi endpoint e all'esperienza dello user ed infine tornare all'applicazione legacy precedente qualora, per qualsiasi ragione, la nuova risulti essere non adatta.

Siccome tutto il traffico passa per l'API Gateway, è possibile ottenere metriche di valore dal servizio, tra cui si citano quelle inerenti al volume del traffico, al tempo di risposta e ai ratei di errore.

In aggiunta è possibile sfruttare il servizio CloudWatch per monitorare le richieste e configurare allarmi e notifiche push.

Considerazioni speciali sulle applicazioni monolitiche

La tecnica illustrata in questo articolo è piuttosto semplice, tuttavia, ci sono alcune considerazioni da fare nel caso in cui si stia cercando di migrare un'applicazione monolitica. Per questa tipologia di applicazioni, infatti, è generalmente necessario prevedere effort extra per "spezzare" il monolita, oltre ad eventuali modifiche all'infrastruttura esistente per trarre il massimo beneficio dai servizi Cloud.

Non sempre è necessario rompere le applicazioni monolitiche per migrarle verso serverless; ad esempio nel caso in cui si stia migrando un back-end che espone semplici API RESTFUL e stateless. In questo caso è possibile percorrere diverse strade tra cui:

- Realizzare un wrapper per inglobare l'intero back-end in una singola lambda function

- Effettuare una riscrittura a medio/basso effort utilizzando un framework come AWS Chalice adatto alla realizzazione di semplici API.

Qualora si intenda apportare modifiche più estese alla codebase ha sicuramente senso considerare una riscrittura volta alla separazione in gruppi di endpoint con funzionalità strettamente correlate, anche chiamati microservizi, ed implementarli in Lambda function separate.

Al crescere della complessità del back-end possono essere impiegati servizi di orchestrazione come AWS Step Functions, servizi di messaggistica (bus, code) come SQS ed Amazon EventBridge.

In caso l'applicazione monolitica comprenda anche un front-end, questo deve generalmente essere separato dal resto dell'applicazione e riscritto per diventare un'applicazione completamente separata. Questo implica generalmente la riscrittura dell'applicazione e la formalizzazione di una API di backend che riesca a soddisfare tutte le necessità dell'applicazione di frontend.

Infine, se l'applicazione mantiene una sessione occorre gestirne il refactoring per permettere al backend di essere sviluppato mediante microservizi. Il modo più semplice ed immediato prevede di spostare lo storage dei dati di sessione all'interno di un database che possa essere raggiunto dalle Lambda functions.

Per lo scopo è possibile considerare di utilizzare un database già utilizzato dall'applicazione, oppure di scegliere una soluzione specifica come ad esempio un in memory DB oppure DynamoDB.

L'operazione di esternalizzazione della sessione può rivelarsi un aspetto oneroso della migrazione, e rappresenta sicuramente una delle prime parti che è possibile realizzare durante la migrazione progressiva.

AWS migration HUB

AWS Migration Hub è uno sportello unico per la migrazione e la modernizzazione del cloud, che fornisce le risorse necessarie per semplificare e accelerare l'utilizzo di AWS.

AWS ha annunciato una nuova funzionalità di Migration Hub durante re:Invent 2021 ed è ora in anteprima: AWS Migration Hub Refactor Spaces. Quest'ultimo è in grado di rifattorizzare le applicazioni esistenti in applicazioni distribuite e cloud-native.

Usando Migration Hub, è possibile sfruttare sfruttare AWS Application Migration Service per semplificare la migrazione e il refactoring di un'applicazione, sia usando il metodo descritto in questo articolo che la strategia di tua scelta.

Conclusioni

Come appena visto, l'approccio serverless permette di migliorare le performance degli applicativi, sia monolitici che a microservizi, garantendo al contempo alta disponibilità ed elasticità.

La migrazione degli applicativi verso un approccio serverless, tuttavia, non è un compito semplice a causa, per esempio, della necessità time consuming di rifattorizzare una parte del codice orginale.

Il servizio API Gateway di AWS risulta essere una soluzione efficiente alle problematiche sopra riportate, in quanto questi può agire come proxy per l'intero applicativo, o altrimenti come unico access point. Agendo da proxy, API Gateway è in grado di gestire efficientemente le richieste ed esegue il routing dell'applicazione legacy con gli endpoint rifattorizzati.

Inoltre, essendo API Gateway il punto di confluenza dell'intero traffico, è possibile ottenere metriche, ad esempio, inerenti al volume dello stesso.

Nell'articolo sono state illustrate anche delle criticità nella migrazione delle applicazioni monolitiche, ma altrettanto è stata proposta una soluzione in grado di risolverle.

Sebbene questa risulti essere una tecnica semplice, siamo convinti che rappresenti un buon punto di inizio per l'elaborazione di strategie complesse e che sia in grado di gestire organicamente la migrazione di applicazioni estese senza causare interruzioni di servizio.

Come siete riusciti a gestire la migrazione di applicazioni verso l'approccio serverless? Fateci sapere nei commenti.

Rimanete collegati per ulteriori interessanti articoli.

Alessio Gandini

Cloud-native Development Line Manager @ beSharp, DevOps Engineer e AWS expert.Computer geek da quando avevo 6 anni, appassionato di informatica ed elettronica a tutto tondo. Ultimamente sto esplorando l'esperienza utente vocale e il mondo dell'IoT.Appassionato di cinema e grande consumatore di serie TV, videogiocatore della domenica.