Multi-Region Deployment: cosa sapere per migliorare affidabilità, disponibilità e sicurezza delle applicazioni

3 Luglio 2024 - 11 min. read

Amazon Cognito | High Availability (HA) | Multi-region deployment

Introduzione

Per fare la differenza nel mercato competitivo odierno, avere un **servizio sempre disponibile** è un aspetto imprescindibile.

Mentre progettare infrastrutture in alta affidabilità è ormai una best practice, l'approccio Multi-Region resta una strada relativamente poco battuta a causa di alcune complessità che la sua implementazione comporta.

Una strategia mutli-region richiede una pianificazione attenta che deve tenerer conto di moltissimi fattori. Lo scenario si complica ulteriormente nel caso di progetti data-driven in cui bisogna prestare particolare attenzione anche a tutti gli aspetti infrastrutturali legati al dato: dal trasferimento, alla gestione, fino all'archivio e all'integrità delle informazioni.

I vantaggi offerti da un approccio multi-region sono moltissimi: distribuendo i dati su diverse posizioni geografiche le applicazioni possono resistere a interruzioni a livello di Region assicurando così **integrità del servizio e continuità operativa**.

Ma non solo: i benefici vanno oltre alta disponibilità, RTO/RPO e resielineza: progettare in Multi-region permette di supportare anche la **scalabilità globale**, consentendo alle applicazioni di servire una base di utenti mondiale con latenza ridotta e prestazioni superiori.

In questo articolo, forniremo una panoramica completa su questo approccio: partiremo dai PRO e i CONTRO di questa strategia e daremo uno sguardo agli aspetti da curare durante

l'implementazione, con particolare attenzione ai database e, in generale, a tutti gli aspetti legati al dato.

Cominciamo!

Quando Serverless e Servizi Gestiti non sono abbastanza

Tipicamente, quando iniziamo un nuovo progetto, cerchiamo di utilizzare solo servizi serverless o gestiti (come AWS Lambda, Amazon APIGateway, Amazon DynamoDB, Amazon Cognito, Amazon SQS e tutti i numerosi servizi gestiti di AWS) pensando di ottenere automaticamente tolleranza ai guasti e alta disponibilità.

E questo è vero; utilizzare servizi serverless ci consente di sfruttare tutte le zone di disponibilità in una regione, rendendo la nostra applicazione resistente ai fallimenti dei singoli data center.

Ma cosa succede se un'intera regione o, più realisticamente, un servizio in tutta la regione, smette di funzionare?

Come possiamo vedere da questo sito che registra tutte le **interruzioni dei servizi AWS** negli ultimi anni, i **fallimenti di una regione**, sebbene molto rari, possono accadere.

Come Solutions Architect, il nostro compito è progettare soluzioni altamente disponibili in termini di zone di disponibilità e creare infrastrutture che **resistano** ai guasti delle regioni.

Questo paradigma non serve solo a creare soluzioni ad alta disponibilità, ma anche a raggiungere utenti in tutto il mondo, avvicinando i contenuti alla loro posizione. In questa discussione, ci concentreremo principalmente sul cambiamento del paradigma per migliorare la resilienza e l'alta disponibilità delle nostre applicazioni.

Tuttavia, anche se utilizziamo solo servizi serverless, cambiare paradigma non è sempre così semplice come sembra.

Intraprendere un cambiamento da mono-region a multi-region comporta alcune **sfide**. Bisogna pensare diversamente e scegliere il servizio appropriato che meglio si adatta alle nostre esigenze e al nuovo paradigma multi-region.

E anche prestando attenzione a tutti questi problemi, a volte ci si accorge che non esiste una soluzione pronta e facile da usare.

Il **data layer** è uno dei primi aspetti da affrontare: ogni servizio memorizza e gestisce i dati con le proprie complessità (e i propri limiti). Per questo bispgna progettare replica e

sincronizzazione dei dati in modo accurato.

Prendiamo poi in considerazione il **layer di autenticazione**: in AWS, l'unico servizio che consente di autenticare le nostre API è Amazon Cognito.

Sfortunatamente, ad oggi questo servizio **non può essere distribuito su piú region** (e non sappiamo se mai sarà possibile farlo). Quindi, come possiamo ottenere l'alta disponibilità in questo caso?

E per quanto riguarda il **logging e il monitoring**?

In una soluzione mono-region, i log e le metriche sono nella stessa regione dell'applicazione e sappiamo per certo che, se un utente avesse dei problemi, potremmo semplicemente consultare i log in AWS Cloudwatch (per esempio) e capire il perché degli errori riscontrati. In una soluzione multi-region, invece, i log e le metriche sono sparsi in più Region. Cercare un errore specifico relativo ad un utente, è quindi più difficile poiché non sappiamo dove si trova l'utente e quale regione sta contattando.

In questo articolo, ci concentreremo su soluzioni per Database, Object Storage, Route53 e sistemi di Cache per trasformare la nostra semplice configurazione mono-region in una soluzione multi-region active-active.

Ma prima, vediamo alcuni vantaggi e svantaggi della scelta di un deployment multi-region.

Deployment Multi-Region: PRO e CONTRO

Come per ogni soluzione, ci sono pro e contro da considerare prima di scegliere ciò che è appropriato per la nostra applicazione. Quindi non perdiamo tempo e iniziamo ad analizzare tutti i vantaggi e gli svantaggi della soluzione Multi-Region - Active / Active.

PRO:

- Come abbiamo già detto, se ci sono problemi in una regione, il nostro servizio **continua a funzionare perfettamente senza il nostro intervento**.
- Il traffico verso la nostra applicazione è **distribuito** tra tutte le regioni configurate, permettendoci di **configurare meglio** tutte le parti computazionali in base al traffico specifico della regione. Questo potrebbe portare a una migliore ottimizzazione dei costi e delle prestazioni.
- Possiamo portare i contenuti più vicino alla posizione dei nostri utenti, offrendo loro migliori prestazioni.

- Per alcuni servizi AWS, esistono limiti regionali non aggiustabili (come l'API GetSecretValue di AWS Secrets Manager che ha un limite non aumentabile di 10.000 richieste al secondo). Avere la stessa risorsa distribuita in diverse regioni, **aumenta** di conseguenza il limite del servizio, permettendoci una migliore scalabilità.

CONTRO:

- Il **logging** e il **monitoring** sono più difficili poiché sono distribuiti in tutte le regioni.
- In base ai servizi utilizzati, i **costi** possono essere più **alti**.
- **La progettazione e l'implementazione** sono più **complesse**.
- L'iinfrastructure as Code (**IaC**) **funziona diversamente**.

Ora che abbiamo visto alcuni pro e contro della soluzione multi-region, andiamo ad approfondire il cuore di questo articolo.

Un'applicazione single-region

Immaginiamo una semplice applicazione in cui dobbiamo gestire delle API CRUD con paradigma REST. Avremo un database con alcune tabelle e un layer API che ci consente di creare, aggiornare, eliminare e ottenere tutti i dati da quelle tabelle.

Una possibile infrastruttura per questo tipo di applicazione è la classica API Gateway / Lambda Function per esporre le API e DynamoDB o RDS Aurora per la parte di database.

Ora che abbiamo definito la nostra infrastruttura di base mono-region, possiamo iniziare a pensare a come riprogettarla nel paradigma multi-region. Per i componenti Amazon APIGateway e AWS Lambda, non ci sono particolari considerazioni. Possiamo semplicemente distribuirli in tutte le regioni scelte.

Il primo oggetto che merita qualche considerazione, quindi, è il nostro **database**. Per evitare fastidiosi problemi di sistema come gli scenari di split-brain, dobbiamo renderlo **globalmente distribuito** in tutte le regioni senza compromettere la struttura dei dati. Per questo, il cloud AWS ci viene in aiuto, mettendo a nostra disposizione **diverse soluzioni** in base al servizio che stiamo utilizzando.

Database

Una delle migliori soluzioni è Amazon DynamoDB.

AWS ha creato una configurazione perfetta di DynamoDB che ci permette di distribuire la stessa tabella in diverse regioni in modo multi-master chiamato **DynamoDB Global Tables**.

Grazie a questo, l'applicazione può leggere e scrivere dati nella tabella regionale, la quale verrá replicata automaticamente in tutte le regioni configurate. È importante ricordare che se diverse istanze scrivono lo stesso record in regioni diverse e nello stesso istante di tempo, AWS utilizza un meccanismo di riconciliazione **last-writer-wins** per salvare e replicare gli **aggiornamenti concorrenti**. Un'altra cosa da sapere è che le scritture transazionali funzionano all'interno della regione e non a livello globale. Il record viene replicato nelle altre regioni solo quando viene salvato nella regione corrente. Quindi, prestate attenzione a questi limiti durante lo sviluppo della vostra applicazione.

Ma cosa succede se abbiamo bisogno di **database relazionali** tradizionali o di altri tipi di database? Fortunatamente, AWS ha creato configurazioni globali per alcuni dei suoi servizi di database, tra cui AWS RDS Aurora e AWS ElastiCache. Ma sono un po' diversi dalla soluzione DynamoDB. Sfortunatamente, non sono in una configurazione multi-master, consistono, invece, in un'istanza master attiva in una regione scelta e diverse istanze di **replica** (in sola lettura) in tutte le altre regioni. In caso di guasti regionali, AWS promuove automaticamente una delle repliche a nodo principale entro pochi minuti. Per soluzioni relazionali multi-region active-active è necessaria una progettazione più complessa, ma oggi ci accontenteremo solo delle configurazioni active-passive.

Con questa soluzione, abbiamo trasformato il nostro database deployato in una singola regione in un database globalmente distribuito che ci permette di salvare e leggere i nostri dati in diverse regioni.

Ora dobbiamo solo instradare il traffico verso la nostra infrastruttura multi-region, e per questo sono necessarie ulteriori considerazioni.

Object Storage

E i dati che non sono memorizzati nel database?

Uno dei metodi di archiviazione dati più comuni è sicuramente quello basato su Amazon S3.

Grazie alla funzionalità Amazon S3 Cross Region Replication (CRR) replicare e sincronizzare i dati su S3 è molto semplice.

CRR copia asincronamente ogni oggetto in uno o più bucket di destinazione in diverse Region di AWS. Il processo di replica include non solo l'oggetto stesso, ma anche i suoi metadati e le access control list conservando le proprietà originali dell'oggetto.

Similmente all'approccio con i database relazionali, anche questo è un sistema di replica primary - standby, quindi con un bucket primario che viene replicato tra le regioni.

Sfortunatamente, l'aggiornamento delle repliche non propaga le modifiche agli altri bucket.

Con questa configurazione possiamo trasformare il nostro database single-region in uno distribuito a livello globale

A questo punto, non ci resta che instradare il traffico verso la nostra infrastruttura multi-region. A questo proposito, vanno fatte ulteriori considerazioni.

Configurazione DNS

Possiamo usare diverse soluzioni. La scelta dipende dagli obiettivi.

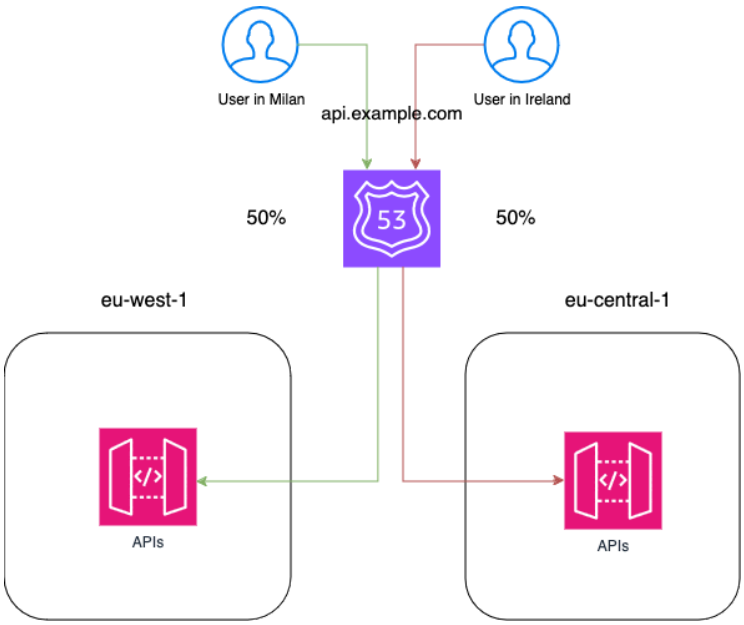
La prima domanda a cui dobbiamo rispondere è: Come vogliamo dividere il traffico nelle regioni configurate? Vogliamo dividere il traffico in base al volume? Prossimità dell'utente? Latenza dell'utente? Disponibilità del servizio?

Approfondiamo alcune di queste soluzioni.

Weighted Records

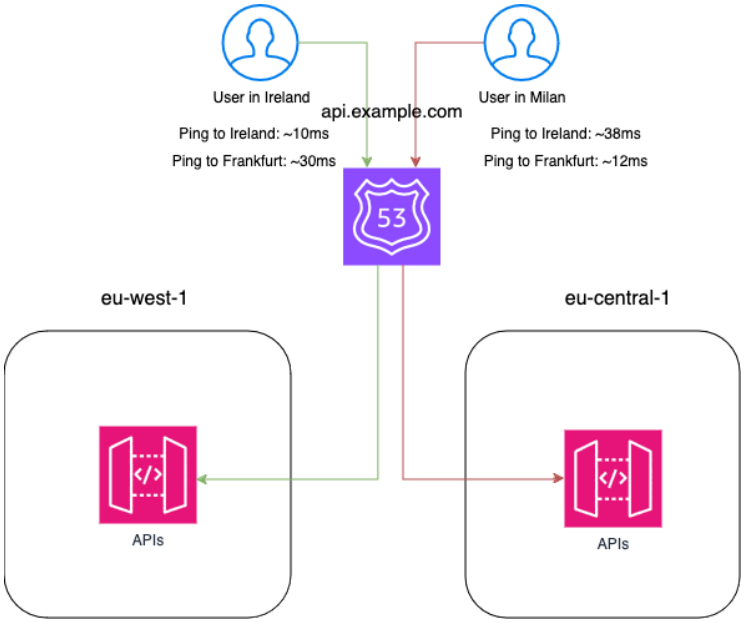
La soluzione più semplice è dividere il traffico bilanciandolo su tutte le regioni disponibili utilizzando i record di tipo **weighted**. Questo ci permetterá di avere circa **lo stesso traffico in entrata** su tutte le regioni.

Supponiamo di aver configurato tre regioni per la nostra applicazione. Possiamo creare tre diversi record di tipo weighted (uno per ciascuna regione) con un peso di 85 (255 come peso massimo diviso sulle 3 regioni). Con questa configurazione, Route53 **bilancerà** automaticamente il traffico in entrata nelle tre regioni.



Latency-Based Records

Un altro modo per configurare il routing DNS è utilizzare i record basati sulla **latenza**. Questo ci permette di instradare il traffico nella regione configurata che è più **vicina** all'utente che sta effettuando la richiesta. Se non abbiamo vincoli applicativi che impediscono questo tipo di configurazione, questa è la migliore soluzione per il routing DNS poichè migliora le prestazioni per l'utente finale.

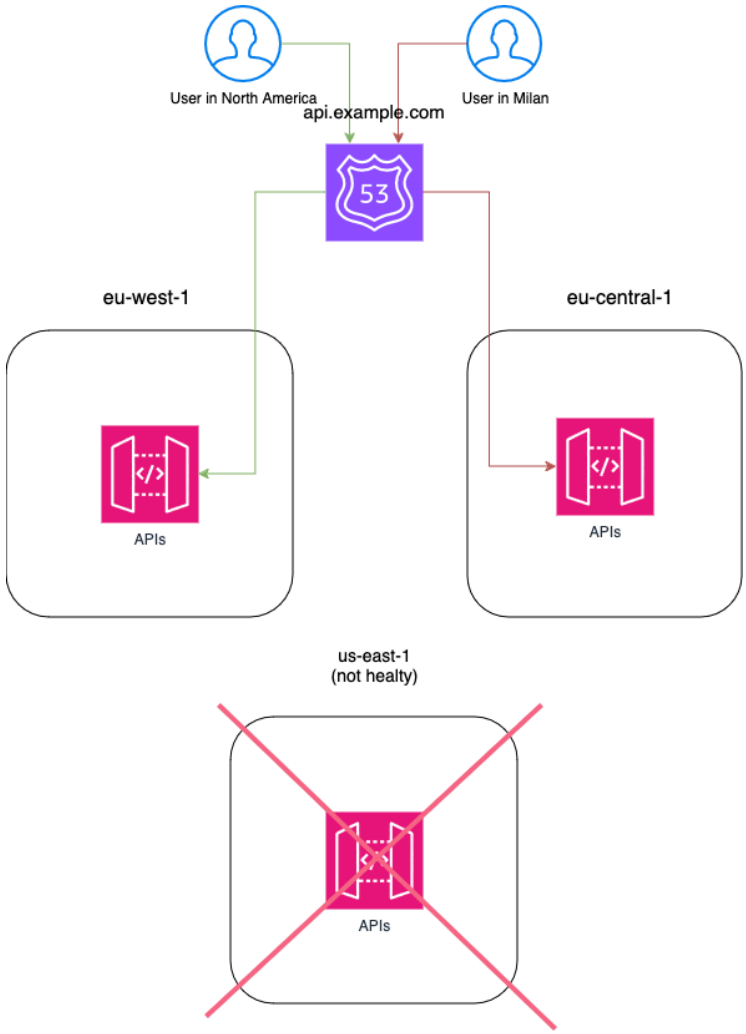


Health Check

Ma cosa succede se una regione ha **interruzioni** e diventa **non piú disponibile?**

Per risolvere questo problema, abbiamo un'altra cosa importante da configurare:l'**health chek**. Grazie ad esso, Route53 sará in grado di instradare il traffico solo verso le regioni funzionanti. Nel nostro caso, poiché stiamo usando API Gateway, possiamo usare i record **alias** che ci permettono di abilitare la funzione "Evaluate Target Health". Con questo flag, Route53 effettua automaticamente un controllo di stato sui nostri servizi e capisce se il target sta funzionando correttamente. Questo check automatico non controlla i codici HTTP o altre metriche ma verifica semplicemente che il servizio sia raggiungibile entro pochi secondi.

Se abbiamo bisogno di usare metriche specifiche o non stiamo usando servizi compatibili con i record alias, possiamo creare i nostri controlli personalizzati. I **Custom health checks** possono verificare il codice HTTP di una specifica API o controllare una metrica specifica di Cloudwatch, permettendoci di **personalizzare** come Route53 dichiara un servizio funzionante.



A questo punto, siamo riusciti con successo a configurare la nostra infrastruttura iniziale in una soluzione multi-region active-active.

Tutto funziona come previsto, ma le lamentele degli utenti sono dietro l'angolo.

Le prestazioni sono cruciali: dobbiamo migliorare l'esperienza degli utenti memorizzando nella cache i risultati delle nostre API in lettura.

Cache Globale

Tipicamente, con API Gateway possiamo configurare una cache che utilizza istanze per memorizzare automaticamente i dati delle API GET. Questo tipo di configurazione ha costi aggiuntivi addebitati in base alle ore di utilizzo e alla dimensione dell'istanza selezionata. In una soluzione multi-region, questo significa costi più alti.

Una semplice soluzione a questo problema è usare Amazon Cloudfront come sistema di cache davanti alla nostra infrastruttura multi-region.

Immaginiamo di voler esporre le nostre API utilizzando il dominio "api.example.com". Possiamo creare la nostra distribuzione Cloudfront e configurarla per memorizzare nella

cache tutti gli endpoint GET. In base ai vincoli della nostra applicazione, possiamo configurare il Time To Live della nostra cache per scadere:

- dopo un certo tempo: i dati verranno automaticamente eliminati dalla cache dopo il tempo configurato.
- mai: possiamo modificare il nostro codice backend per creare una richiesta di invalidazione della cache di CloudFront sul percorso appropriato solo quando i dati cambiano.

Poi, come origine, useremo l'integrazione HTTP utilizzando il dominio "regional.example.com", che è collegato ai nostri ApiGateway con una delle configurazioni discusse in precedenza. E questo è tutto. Ora, abbiamo un sistema di cache davanti alla nostra infrastruttura.

Conclusioni

In questo articolo, abbiamo visto quali passi dobbiamo compiere per trasformare una semplice soluzione che utilizza una singola Region, in una soluzione multi-region active-active.

L'obiettivo non era solo mostrarvi come possiamo trasformare la nostra applicazione mono-region in una globalmente distribuita, ma anche farvi riflettere su quanti aspetti debbano essere presi in considerazione prima di iniziare a pensare a questo tipo di approccio. Un suggerimento è pensarci prima dell'implementazione effettiva, cambiando il paradigma "from mono-region to multi-region" a "Go Global" direttamente.

Specialmente in un mondo data-driven, essere sicuri che i nostri dati siani sempre aggiornati e accessibili, anche in caso di guasti a livello di region, è una priorità, sia per garantire continuità operativa, che per ottenere vantaggi competitivi.

Ma c'è molto altro da dire quando si parla del paradigma "Go Global". Come menzionato nell'introduzione, durante l'implementazione possono verificarsi diversi problemi:

- Come risolviamo il problema dell'autenticazione?
- Come monitoriamo la nostra soluzione globale?
- Come centralizziamo i log?

...e molti altri che approfondiremo nei prossimi articoli.

Quale argomento vorreste affrontare per primo?

Faccelo sapere nei commenti!

About Proud2beCloud

Proud2beCloud è il blog di beSharp, APN Premier Consulting Partner italiano esperto nella progettazione, implementazione e gestione di infrastrutture Cloud complesse e servizi AWS avanzati. Prima di essere scrittori, siamo Solutions Architect che, dal 2007, lavorano quotidianamente con i servizi AWS. Siamo innovatori alla costante ricerca della soluzione più all'avanguardia per noi e per i nostri clienti. Su Proud2beCloud condividiamo regolarmente i nostri migliori spunti con chi come noi, per lavoro o per passione, lavora con il Cloud di AWS. Partecipa alla discussione!

Mario Agati

DevOps Engineer e Backend developer @ beSharp. Appassionato di musica, calcio e motori, nel tempo libero mi piace dare fastidio ai miei vicini suonando la batteria.
