VPC Lattice: l'ennesima service mesh o un game changer?

12 Aprile 2024 - 6 min. read

Advanced Networking ***VPC Lattice***

"Vivi come se dovessi morire domani. Impara come se dovessi vivere per sempre."

– Mahatma Gandhi

Mi piace imparare cose nuove e sono fortunato perché una parte significativa del mio lavoro comporta la valutazione di nuove tecnologie e l'approfondimento di come utilizzarle per risolvere problemi sia nuovi, che esistenti.

Negli ultimi anni, abbiamo visto molte opzioni per collegare i nostri workload su AWS (Transit Gateway, Load balancing, Shared VPCs, PrivateLink, Endpoint Services, VPC peering...)

L'anno scorso, poi, VPC Lattice è stato rilasciato in General Availability, offrendoci una nuova strada da esplorare. Vi starete chiedendo perché dovremmo imparare come utilizzare l'ennesimo servizio quando abbiamo già così tante opzioni tra cui scegliere... beh, scopriamolo!

In questo articolo, capiremo perché avevamo davvero bisogno di aggiungere un'altra freccia al nostro arco. Vedremo cosa offre questo servizio e come si differenzia dalle altre opzioni di connettività che già conosciamo, concludendo con uno use case di esempio.

Senza ulteriori indugi, cominciamo!

Dove si colloca VPC Lattice

Come abbiamo visto negli articoli precedenti, quello a microservizi è un approccio architetturale che permette agli sviluppatori di costruire e distribuire software più velocemente.

Tuttavia, questo approccio introduce delle sfide: le singole applicazioni sono suddivise in numerosi componenti singoli che possono utilizzare tecnologie differenti (come Lambda, EKS, ECS ed EC2) e che devono comunicare tra loro, anche se distribuiti in diversi account AWS.

Dobbiamo affrontare tematiche già note, come il service discovery, il routing del traffico, l'autorizzazione, la sicurezza, e l'osservabilità.

Ci sono scenari in cui una strategia per far comunicare microservizi in modo sicuro non è facilmente realizzabile con approcci di rete "classici".

Pensate ad esempio a come realizzare un meccanismo di autenticazione per permettere la comunicazione tra due microservizi in diversi account AWS, bloccando allo stesso tempo il traffico estraneo all'applicazione.

VPC Lattice entra in gioco per permetterci di vincere questo tipo di sfide.

Il suo scopo è aiutare le organizzazioni a superare le sfide legate alla gestione sicura dei microservizi, consentendo ai team di sviluppo la definizione dei microservizi e garantendo al contempo agli amministratori la gestione dell'infrastruttura, delle politiche di comunicazione e della governance.

Componenti di VPC Lattice

Vediamo le componenti-chiave di questo servizio:

- **Service**: rappresenta un'applicazione e le sue risorse di calcolo (come ad esempio, Lambda, pod EKS, istanze EC2), listener (porte e protocollo) e regole di routing per indirizzare il traffico (routing pesato, path based, ecc.). Il team può gestire questi aspetti e definirli autonomamente. Un servizio può essere condiviso con altri account AWS utilizzando AWS Resource Access Manager, anche al di fuori dell'organizzazione.
- **Service Network**, un nuovo concetto: si tratta di un raggruppamento logico di servizi, connessi fra loro. Anche la service network può essere condivisa usando AWS Resource Access Manager, viene definita dagli amministratori di sistema.

- **Service Directory**: si tratta dell'elenco delle reti di servizi che possono essere utilizzati e condivisi tramite AWS RAM.
- **Auth Policies**: sono documenti IAM che definiscono l'accesso alle risorse. Differiscono dalle policy IAM perché non sono associate a utenti, gruppi o ruoli, ma vengono utilizzate con servizi e reti di servizi. Possono consentire l'accesso ad una risorsa non solo ad un utente/ruolo IAM, ma anche ad altri servizi VPC Lattice. Per questo sono definite nuove condizioni specifiche, come ad esempio

```
vpc-lattice-svcs:RequestMethod
```

La lista completa è disponibile qui.

Configurazione di VPC Lattice

Vediamo ora quali passaggi sono necessari e i ruoli coinvolti nella configurazione di VPC Lattice:

1. L'amministratore definisce una **Service Network**.
2. L'amministratore condivide la **Service Network** utilizzando **AWS RAM**.
3. Il proprietario del servizio definisce un **VPC Lattice Service** e le relative policy di autorizzazione.
4. Il proprietario del servizio associa il servizio alla **Service Network**.
5. L'amministratore dell'infrastruttura associa la **Service Network** alle VPC.

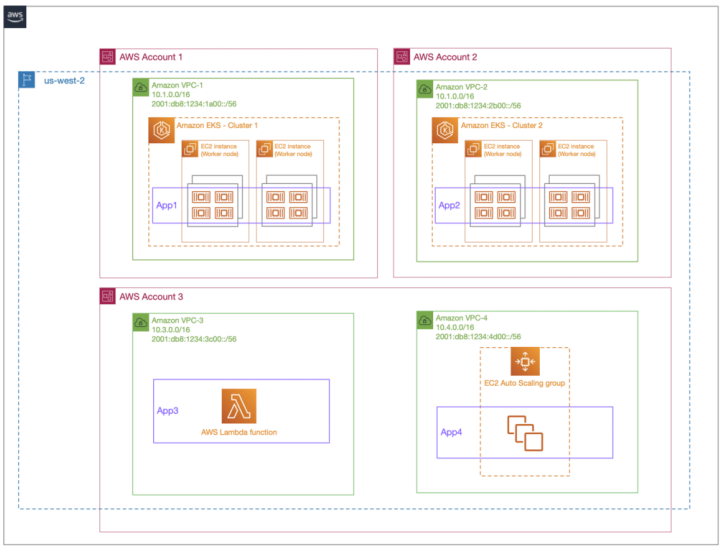
Usando questo tipo di approccio le dipendenze fra il team di sviluppo ed il team di operations si riducono.

Esempio Pratico di Utilizzo di VPC Lattice

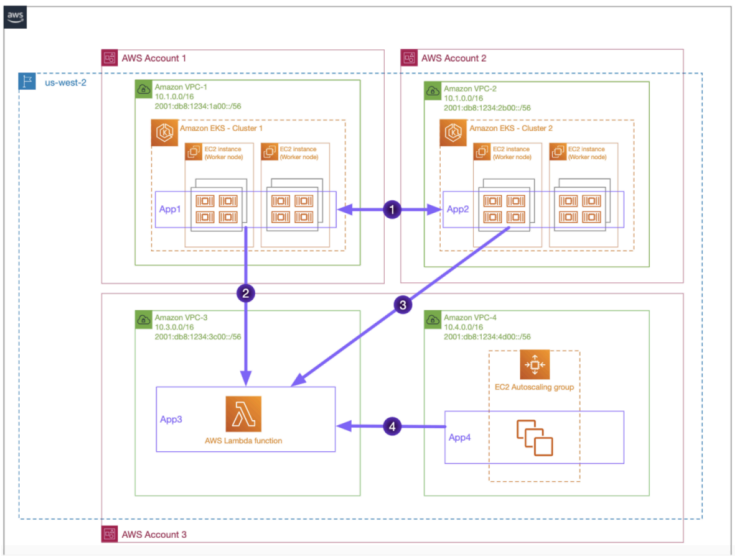
Nel nostro esempio, faremo deploy dell'infrastruttura e delle applicazioni di esempio disponibili su GitHub per analizzare i componenti chiave implementati sulla Console AWS.

Per effettuare il deploy della soluzione basta seguire le istruzioni contenute nel file *ReadMe*. Per semplicità è possibile usare un singolo account AWS con VPC multiple (come fatto da noi in questo articolo).

L'infrastruttura risultante sarà questa:



App1 ed App2 dovranno poter comunicare fra loro in maniera bidirezionale, mentre App3 sarà utilizzata da App1, App2 e App4, con questo flusso comunicativo:



Il deployment richiede circa 10-15 minuti.

Attenti al bug!

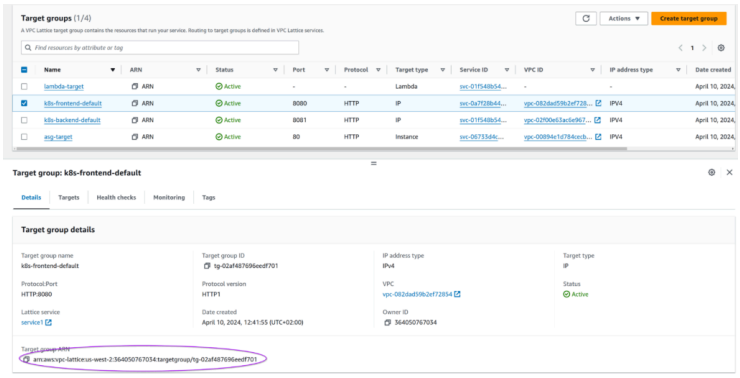
Nella documentazione relativa al deploy manca un passo: queste due istruzioni faranno fallire il deploy dell'ultimo stack CloudFormation

```
export TARGETCLUSTER1={TARGET_GROUP_ARN}
export TARGETCLUSTER2={TARGET_GROUP_ARN}
```

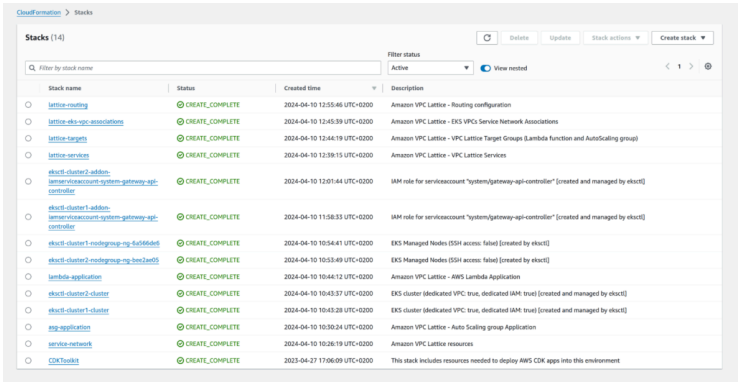
Occorre cambiare i valori esportati con i valori relativi ai VPC Lattice Target Group, andando alla sezione della console AWS "VPC Lattice -> Target Groups".

Per TARGETCLUSTER1 occorre utilizzare l'arn di k8s-frontend-default, per TARGETCLUSTER2 l'arn di k8s-backend-default.

Questo l'esempio della nostra configurazione:

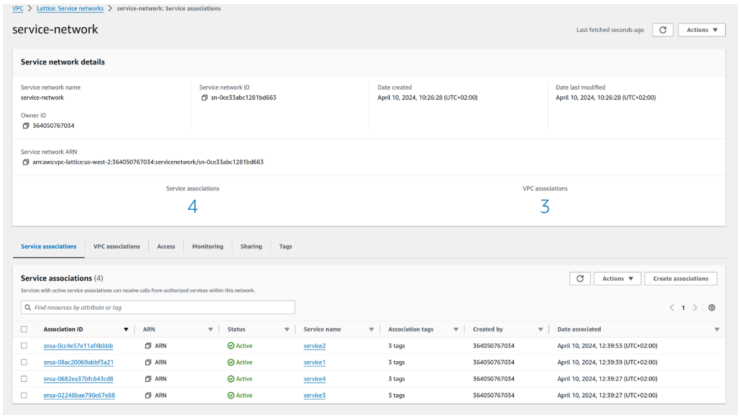


Al termine di tutti i deploy, dovrebbero essere presenti questi stack:

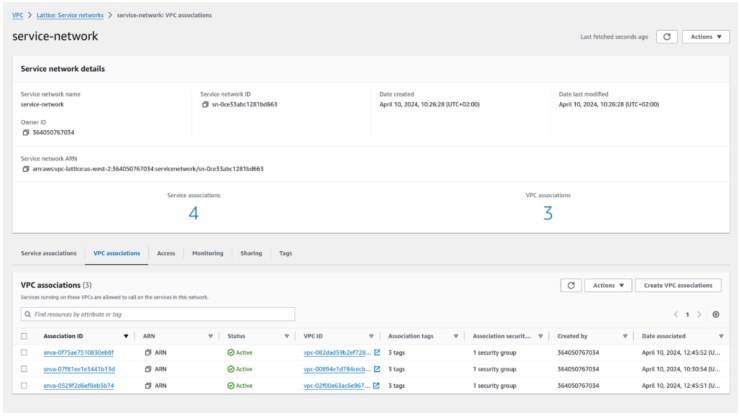


Passiamo ora a vedere finalmente le risorse create.

Per prima cosa la service network:

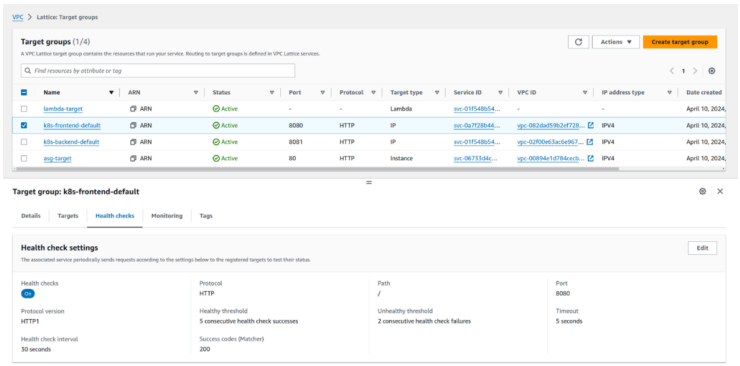


Sono associati 4 servizi e 3 VPC



VPC Lattice Target Groups:

Le nostre Lambda, EKS, e l'autoscaling group sono aggiunti come target. Analogamente a quel che accade per ELB, è possibile vedere le metriche e gli health check.

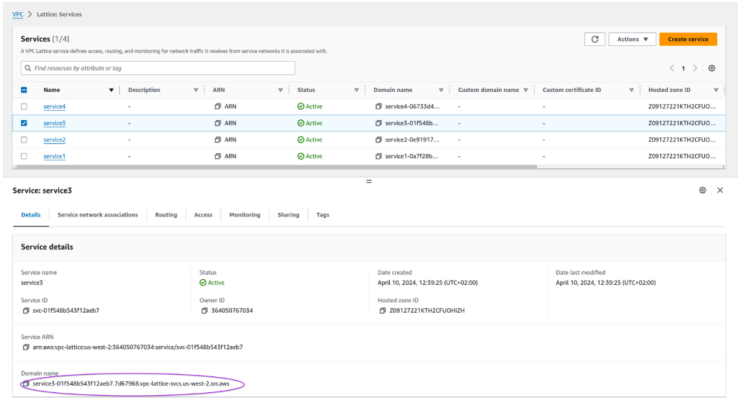


Nota: per registrare come target i servizi gestiti dal cluster EKS, è necessario fare il deploy della class Gateway API Controller nel cluster e registrare il servizio facendo deploy con i comandi:

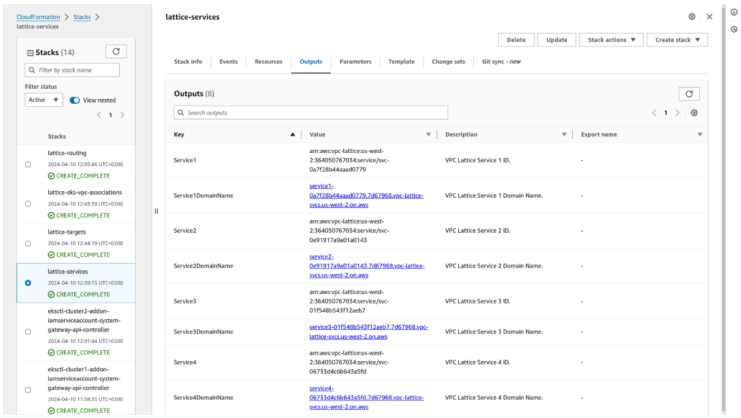
```
kubectl config use-context cluster1
kubectl apply -f ./vpc-lattice/routes/frontend-export.yaml

kubectl config use-context cluster2
kubectl apply -f ./vpc-lattice/routes/backend-export.yaml
```

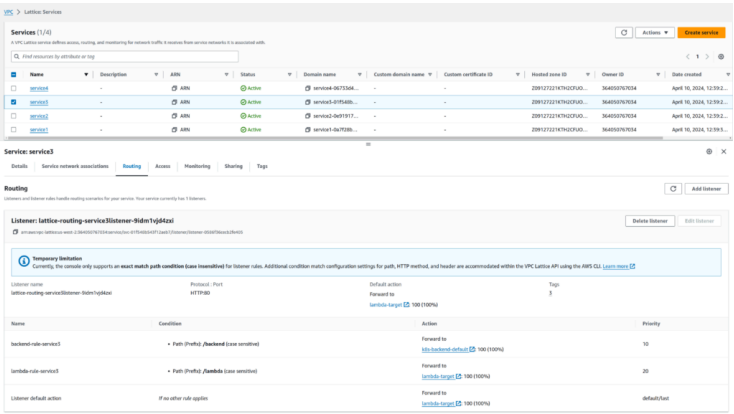
Per ultimi, ma non meno importanti, vediamo i servizi definiti. Sono anche accessibili utilizzando il nome DNS definito da Lattice:



Lo stack CloudFormation "lattice-services", che si occupa della creazione dei servizi, fa in modo che i nomi generati siano riassunti alla sezione "Output" dello stack



Per ogni servizio è possibile definire il routing applicativo. In questo esempio possiamo vedere che la rotta /backend gira le richieste a Cluster2, mentre /lambda fa il forward alla applicazione Lambda:



A questo punto avete una panoramica dell'ambiente. Potete sperimentare e modificare la configurazione di applicazione e ambiente (ad esempio, perché non provare l'autenticazione IAM fra servizi?).

Questo laboratorio è solo il punto di partenza; potete trovare anche un ottimo workshop qui.

Infine: VPC Lattice fa al caso mio?

Come al solito, la risposta è "dipende". Parliamo brevemente di limiti e impatti sul design delle nostre applicazioni.

Il primo limite è dovuto al fatto che l'**associazione fra VPC e Service Network è univoca**: non è infatti possibile associare una VPC a pìu service network. In questo caso il design e la pianificazione dei deployment devono essere valutati attentamenti.

Non è possibile condividere servizi fra region differenti: **la Service Network è una risorsa regional**, quindi in questo caso dovremo usare un approccio ibrido ricorrendo ad opzioni di connettività più tradizionali come il peering fra Transit Gateway e il VPC Peering.

Oltre ai limiti, ovviamente, ci sono benefici: la configurazione del servizio può essere delegata agli owner, mantenendo però un controllo centralizzato, nonostante l'ambiente connesso.

Non abbiamo trattato le policy IAM, ma implementarle permette di avere una architettura a microservizi veramente sicura.

VPC Lattice offre anche una soluzione completa per l'implementazione di una service mesh perché può mettere in comunicazione workload basati su EC2, ECS, EKS e Lambda.

Il costo, d'altro canto, è un fattore importante: il billing cresce infatti al crescere della service mesh.

Per ogni servizio viene addebitato un costo di 0,25$/ora. Con 10 microservizi la bolletta mensile sarà quindi di 182,5$. Per progetti con centinaia di microservizi utilizzare approcci differenti potrebbe avere un impatto minore sul billing.

Per concludere...

Una architettura a microservizi non è una cosa semplice: la complessià aumenta, ed è richiesto un costante sforzo di pianificazione e manutenzione durante tutta la vita del progetto.

Avete già sperimentato l'utilizzo di VPC Lattice?

Fateci sapere la vostra opinione su questo nuovo tipo di approccio nei commenti!

Damiano Giorgi

Ex sistemista on-prem, pigro e incline all'automazione di task noiosi. Alla ricerca costante di novità tecnologiche e quindi passato al cloud per trovare nuovi stimoli.L'unico hardware a cui mi dedico ora è quello del mio basso; se non mi trovate in ufficio o in sala prove provate al pub o in qualche aeroporto!
