Best practices per il logging su AWS: lo stack EKK.

28 Maggio 2021 - 6 min. read

Amazon Elasticsearch Service | Amazon S3 | Kibana | Logging | Logstash

Introduzione

Oggigiorno è sempre più importante poter monitorare e tracciare lo stato delle proprie applicazione, come saper identificare facilmente la fonte di problematiche. Contando il numero sempre più crescente di servizi digitali, indipendentemente dalla loro grandezza e importantanza, questa esigenza è sempre più sentita.

Ad oggi ci si scontra anche con molti pattern infrastrutturali moderni e sempre più complessi, pensiamo al mondo microservizi o serverless. Bisogna trovare strumenti efficaci e centralizzati per il monitoraggio.

Lungi da noi andare a descrivere e comparare i vari programmi di logging management, non basterebbe un solo articolo! Possiamo dire, però, che sul mondo AWS questi problemi vengono notevolmente ridotti grazie alle numerose alternative che abbiamo a disposizione, unitamente al vantaggio dei servizi totalmente gestiti!

In questo articolo andremo a parlare di come centralizzare e gestire in modo efficiente i log provenienti da varie applicazioni, restando interamente sul mondo AWS! Nello specifico, esploreremo un'alternativa alla popolare soluzione di log aggregation, lo stack ELK (Elasticsearch, Logstash, Kibana), ovvero lo stack EKK (Amazon Elasticsearch Service, Amazon Kinesis e Kibana).

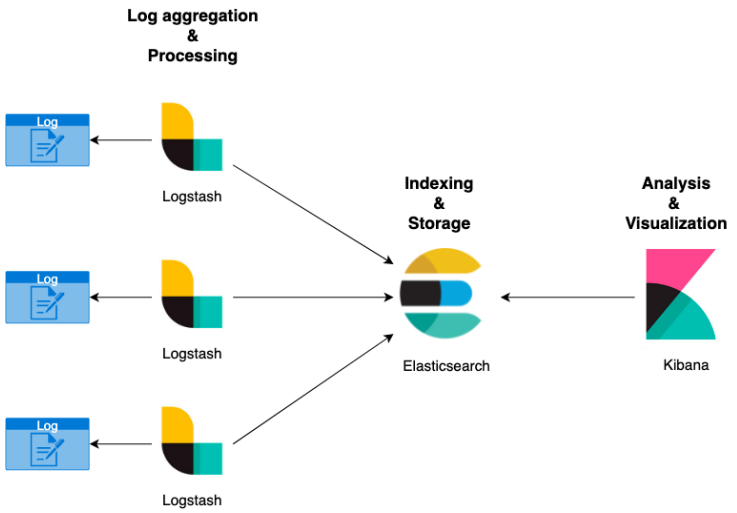
Lo stack EKK

Iniziamo con una breve descrizione dello stack ELK. Per chi di voi lo conoscesse, facciamo un ripasso insieme! Come per la storia, anche nell'informatica risulta utile

conoscere il passato per comprendere a meglio il presente.

Partendo dal diagramma riportato sotto, lo stack ELK è composto dai seguenti componenti:

- **Logstash**: Strumento di aggregazione, eventuale processing e inoltro di log da un sistema *source* verso un server Elasticsearch. Questo componente si occupa inoltre di funzioni come il retry, il batching e l'encryption dei log prima di essere mandati al server.
- **Elasticsearch**: E' un server di ricerca, basato su Lucene, che è predisposto alla ricezione dei log per salvarli sotto *indici*. Particolare attenzione è da prestare in fase di setup per garantire l'alta affidabilità, oltre alla scelta opportuna degli indici. Tali tematiche non verranno trattate in questo blog.
- **Kibana**: Interfaccia grafica per interrogare Elasticsearch. Svolge lei il "lavoro sporco" di interrogare il server tramite API e di presentarci graficamente i log con dashoard e filtri di ricerca per contenuto e data.



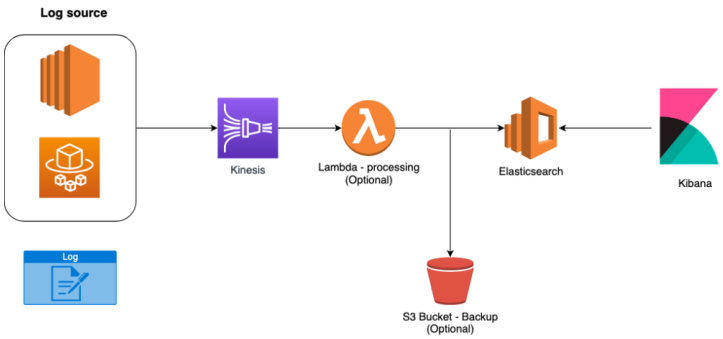
Presentato così, potrebbe risultare relativamente complessa la semplice funzione di "logging" della nostra applicazione. Tuttavia, questo stack ad oggi è utilizzato moltissimo per la sua versatilità e la sua scalabilità.

Ma non è di questa struttura che andremo a parlare, bensì di un suo "fork" cloud-native sul mondo AWS! Tratteremo infatti lo stack EKK, i componenti che entrano in gioco diventano:

- **Amazon Elaticsearch service:** il servizio di AWS che supporta una versione open source di Elasticsearch (Open Distro for Elasticsearch)

- **Amazon Kinesis Firehose**: un servizio di stream affidabile, robusto e near real-time, usato per scenari come data lake o acquisizione di video, audio e clickstream
- **Kibana**

Il diagramma infrastrutturale si trasforma come di seguito:



Comodo, vero? Dimentichiamoci il provisioning e la gestione di macchine EC2, oltre al loro scaling, manutenzione e configurazione in alta disponibilità. Questi servizi, totalmente gestiti da AWS, vengono in nostro soccorso permettendoci di concentrarci sulla nostra applicazione e ridurre attività costose, in termini di tempo, per la "banale" infrastruttura legata all'aggregazione e presentazione dei log.

Per i più diffidenti, il dover mandare i dati a Firehose potrebbe non risultare la scelta migliore. Ma vediamo di illustrarne i vantaggi.

Innanzitutto, esiste un'integrazione nativa tra Kinesis e Elasticsearch gestito da AWS! Inoltre, possiamo decidere di effettuare attività di processing su tutti i dati in ingresso, se fosse ad esempio necessario formattare i log in arrivo da tutti i sistemi sorgenti.

Non basta? Su Kinesis possiamo anche selezionare un bucket S3 per salvare i log, così facendo saranno disponibili in futuro per eventuali analisi o spostamento su altre piattaforme.

Per ridurre i costi, nulla ci vieta di spostare i dati su altri S3 Tier più economici rispetto a quello Standard.

Tra le funzioni offerta da Kinesis, c'è poi quella di batching, encryption e retry per gestire al meglio le richieste verso il server di Elasticsearch.

Ma quali sorgenti possiamo utilizzare con Kinesis? Idealmente, qualsiasi! E' infatti possibile richiamare direttamente le API (ad esempio quelle fornite dai vari SDK), oppure possiamo utilizzare strumenti già forniti da AWS se stiamo utilizzando i suoi servizi di computing più noti.

Facciamo qualche esempio:

- **EC2**: Grazie a *Amazon Kinesis Agent*, un software Java-based che raccoglie e invia i log in autonomia a Kinesis
- **ECS Fargate**: L'integrazione con Kinesis può essere effettuata tramite il log driver *FireLens*
- **Servizio di altro tipo**: Richiamando direttamente le API di Kinesis.

Riassumendo, abbiamo visto come riportare il classico stack ELK in ottica cloud-native AWS, trasformandolo quindi in stack EKK per sfruttare i vantaggi dei servizi gestiti da AWS.

Non vedete l'ora di fare un po' di pratica? Perfetto! Facciamo una breve demo utilizzando un cluster ECS in modalità Fargate!

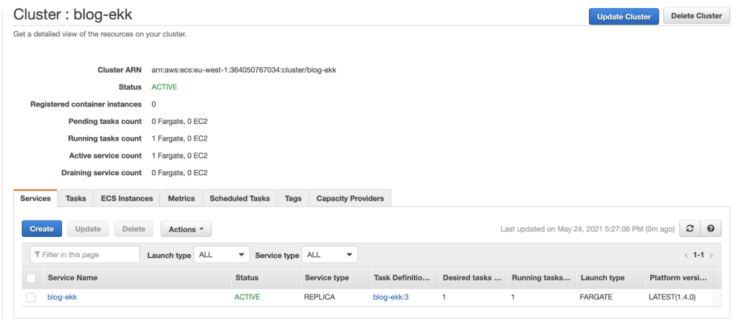
EKK su ECS Fargate

Prima di partire con la demo, elenchiamo i servizi che andremo ad utilizzare:

- **ECS Cluster Fargate**: Abbiamo già visto ampiamente in questo blog come crearlo e configurare eventuali service e task.
- **Elasticsearch domain:** Non tratteremo il dettaglio della sua configurazione. Il consiglio che posso darvi è di prestare attenzione ai seguenti aspetti:
  - Dimensionamento della parte computazionale, eventuale alta disponibilità, dimensionamento dello storage opportuno
  - Gestire i pattern di accesso al server elasticsearch (assimilabili a resource-policy)
  - Gestire l'accesso a Kibana opportunamente, nel nostro caso abbiamo utilizzato un Cognito User Pool insieme a Cognito Identity Pool
  - Configurazione di rete: Questa parte non sarà più modificabile dopo la creazione. Nel nostro caso, abbiamo scelto un'istanza con accesso su internet pubblico.

- **Kinesis Firehose**: Come da diagramma infrastrutturale, il servizio di AWS verrà utilizzato per l'ingestion dei dati.

Ecco il nostro cluster Fargate con il *service* creato per la demo, *blog-ekk*:



Particolare attenzione è da porre nel definizione del *task definiton*, andremo infatti a utilizzare *FireLens* per l'integrazione nativa tra Fargate e Firehose:

FireLens è un log driver per i container ospitati su Amazon ECS che estende le sue funzionalità per la gestione dei log. Si appoggia a Fleuntd e Fluent Bit.

Di seguito viene riportata una semplificazione del task definition utilizzato da noi:

{ "family": "firelens-example-firehose", "taskRoleArn": "arn:aws:iam::XXXXXXXXXXXX:role/ecs_task_iam_role", "executionRoleArn": "arn:aws:iam::XXXXXXXXXXXX:role/ecs_task_execution_role", "containerDefinitions": [{ "essential": true, "image": "amazon/aws-for-fluent-bit:latest", "name": "log_router", "firelensConfiguration": { "type": "fluentbit" }, "logConfiguration": { "logDriver": "awslogs", "options": { "awslogs-group": "firelens-container", "awslogs-region": "eu-west-1", "awslogs-create-group": "true", "awslogs-stream-prefix": "firelens" } }, "memoryReservation": 50 }, { "essential": true, "image": "your-image-uri", "name": "app", "logConfiguration": { "logDriver":"awsfirelens", "options": { "Name": "firehose", "region": "eu-west-1"", "delivery_stream": "blog-ekk" } }, "memoryReservation": 100 }] }

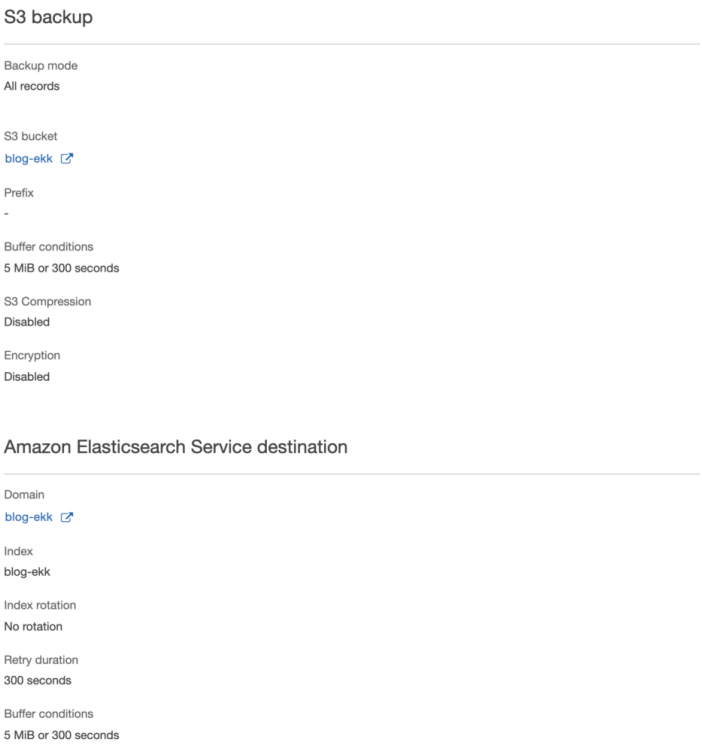
Se utilizziamo l'interfaccia grafica per la creazione del nostro task definition, una volta configurato il nostro container, modificando il log router in firelens apparirà in automatico il side container che si appoggia a FluentBit (container image: *amazon / aws-for-fluent -bit: latest*)

Non dimenticatevi di cambiare il task definition role affinché abbia i permessi di scrivere su Firehose!

Perfetto, il sistema sender è stato configurato! Passiamo ora alla definizione del sistema di ingestion: Kinesis Firehose!

La sua configurazione risulta relativamente semplice. Nel nostro caso abbiamo utilizzato S3 come backup, Elasticsearch come destinazione e nessuna trasformazione

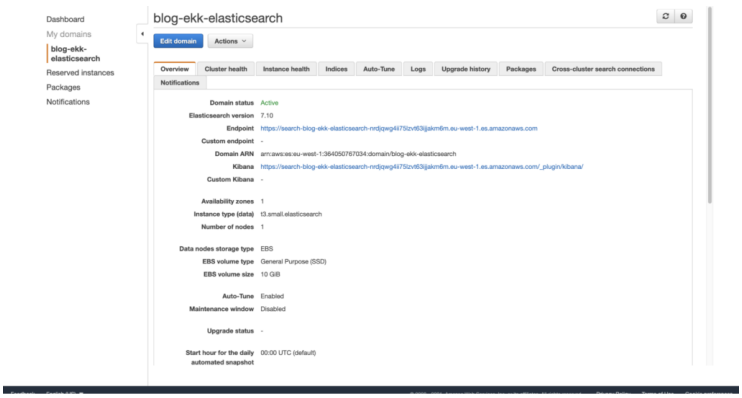
Lambda.



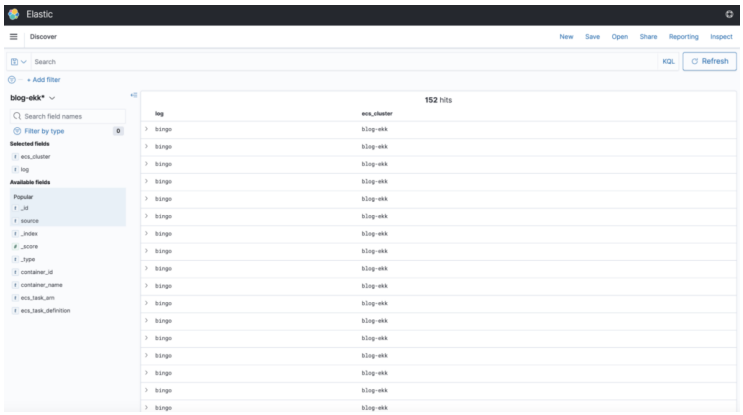
Avete configurato tutto? Assicuratevi che il vostro Service su ECS sia up e running, nella sezione *log* potete controllare che il side container stia effettivamente inoltrato i log a Kinesis.

Per ulteriore verifica, potete controllare dalle metriche su Kinesis la corretta ricezione e il corretto inoltro sul server Elasticsearch.

Non ci resta che entrare sul dominio di Elasticsearch, nella pagina di overview troveremo l'indirizzo per accedere a Kibana. Nel nostro caso l'accesso verrà autenticato tramite username e password salvati e gestiti da Cognito:



Ed ecco qui la nostra dashboard su Kibana:



Il comportamento è quello atteso, abbiamo infatti creato un semplice container che scrive “bingo” periodicamente tramite un setInterval in Javascript.

Conclusioni

Per concludere, in questo articolo abbiamo descritto i vantaggi dello stack EKK, una versione cloud-native AWS del classico stack ELK.

Abbiamo poi visto quali possibili servizi possiamo utilizzare per inoltrare i nostri log, approfondendo il caso di container ospitati su ECS in modalità Fargate.

Volete approfondire questa modalità con una guida passo-passo? O magari siete curiosi della soluzione partendo da macchine EC2? Write us in the comments, and we will try to accommodate your requests!

Alessandro Bertini

DevOps Engineer @ beSharp, mi occupo di sviluppo software Cloud-native, fortemente orientato al paradigma Serverless!Appassionato di giochi da tavolo e videogame (come ogni buon smanettone!)
