OpenSearch: Guida completa alla configurazione perfetta e al suo utilizzo su AWS

4 Agosto 2023 - 10 min. read

Data and Analytics | Data Visualization | OpenSearch

In un mondo in cui **avere un approccio basato sui dati** sembra essere l'unica via per rimanere competitivi in un mercato in rapida evoluzione, la gestione efficace e l'elaborazione real-time di grandi quantità di dati, insieme a sistemi efficaci di visualizzazione sono sempre più una priorità per le aziende di ogni settore e dimensione.

In questo contesto, la scelta dello strumento che meglio soddisfa le esigenze aziendali sulla base di quantità e tipologia di informazioni da analizzare è spesso meno semplice di quanto sembri.

In questo articolo, presentiamo OpenSearch, un set di strumenti altamente scalabile in grado di fornire accesso rapido a grandi volumi di dati.

Cominciamo!

Terminologia

OpenSearch è una suite di ricerca e analisi open source utilizzata per eseguire query su grandi volumi di dati utilizzando chiamate API o un dashboard integrata. OpenSearch offre funzionalità come query full-text, completamento automatico, ricerca a scorrimento, punteggio e classificazione personalizzabili, ricerca *fuzzy*, corrispondenza di frasi e altro ancora. Le risposte possono essere restituite in formato jdbc, csv, raw o JSON.

Facciamo una breve descrizione dei componenti fondamentali di un cluster OpenSearch:

- Indici

- Shards
- Nodi
- Tipi di nodo

Per cercare i dati è necessario organizzarli in **indici**. Gli indici memorizzano i documenti (insiemi di campi con coppie chiave-valore) e li ottimizzano. L'ottimizzazione è possibile perché ogni campo ha un tipo specifico. È possibile specificare i tipi di campo. Se questo non avviene, sarà OpenSearch a tentare di determinare automaticamente il tipo.

Un'altra forma di ottimizzazione è la suddivisione dell'indice in diversi **shards**. Ciascun frammento contiene un sottoinsieme dei documenti all'interno dell'indice. Quando si cercano dei dati, le query vengono eseguite su diversi shard in parallelo se ogni shard si trova su un nodo diverso. La dimensione degli shard dovrebbe essere di circa 10-30 GB per carichi di lavoro che richiedono una bassa latenza di ricerca e 30-50 GB per carichi di lavoro pesanti in scrittura come l'archiviazione dei log.

Le istanze di OpenSearch sono chiamate **nodi**. OpenSearch può funzionare come un cluster a nodo singolo o multi-nodo. In quest'ultimo caso, il numero di nodi, i tipi di nodo e il relativo hardware dipendono dal caso d'uso.

I **tipi di nodi** sono:

- **Master**: il nodo master gestisce attività come la gestione degli indici, il monitoraggio dei nodi del cluster, l'esecuzione di controlli di integrità e l'allocazione di frammenti
- **Master-eleggibili**: i nodi master-eleggibili possono essere promossi a master attraverso un processo di votazione
- **Data**: i nodi di dati eseguono operazioni relative a tutti i dati su shard locali come l'indicizzazione, la ricerca e l'aggregazione
- **Ingest**: I nodi ingest eseguono le pipeline per trasformare i dati prima di memorizzarli
- **Coordinamento**: i nodi di coordinamento delegano le richieste del client ai nodi di dati e aggregano i risultati in uno prima di inviarlo al client.

Un nodo può avere più tipi. Per impostazione predefinita, ogni nodo è un nodo master di dati, di acquisizione e di coordinamento.

Durante la creazione di un cluster AWS OpenSearch, puoi personalizzare i nodi di dati e i nodi master dedicati.

Per i nodi di dati, puoi specificare il numero di nodi, il tipo di istanza, il tipo di volume e la dimensione del volume.

I Master Node dedicati sono nodi che non contengono dati e sono dedicati alla gestione del cluster. Puoi scegliere di non avere alcun nodo master dedicato negli ambienti di sviluppo o test. Poiché non contengono dati, è necessario specificare solo il numero di nodi e il tipo di istanza.

Tipologie di provisioning

Su AWS è possibile creare un cluster OpenSearch in due modi diversi:

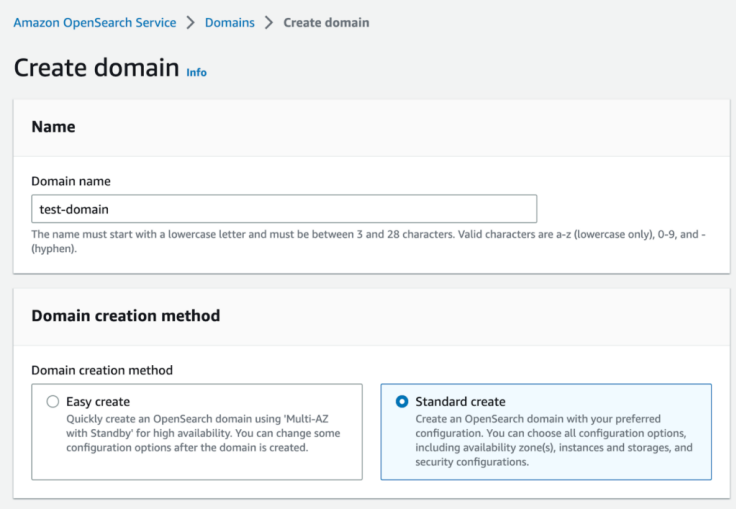
- Serverfull
- Serverless

Serverfull

L'approccio classico è quello **Serverfull** in cui occorre scegliere quanti nodi si desidera creare, specificandone il tipo, il dimensionamento e altre proprietà.

Diamo un'occhiata alla console AWS e analizziamo il **servizio Amazon OpenSearch**. Il primo passo che faremo sarà la **creazione di un dominio**; in altre parole un cluster OpenSearch.

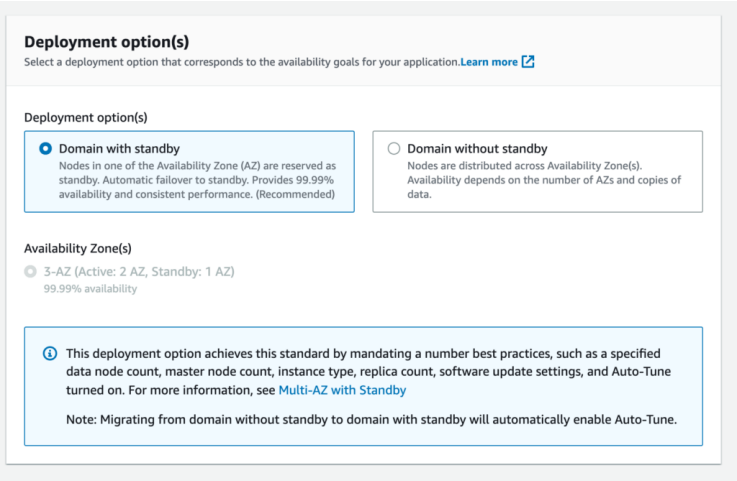
Potrai scegliere tra due opzioni: **Easy create** o **Standard create**.



Il nostro consiglio è quello di scegliere l'opzione di **Standard Create** poiché è più flessibile. L'opzione di **Easy create** è consigliata invece per brevi demo o POC.

Optando per la **Standard create**, ci verrà chiesto di scegliere tra due template già pronti: **Produzione** o **sviluppo/test**. Suggeriamo di saltare questa scelta e di continuare con la configurazione personalizzata.

La selezione più importante è il tipo di ridondanza del cluster:



È importante prestare attenzione a queste opzioni poiché con queste scelte potrebbe cambiare drasticamente il prezzo che andremo a pagare.

Infatti, l'opzione **Dominio con standby** creerà un cluster con minimo 3 nodi di dati, ed sarà possibile incrementarli solo di multipli di 3.

Con il **Dominio senza standby**, è possibile personalizzare il numero di nodi, ma nessuno di essi sarà riservato come nodo standby.

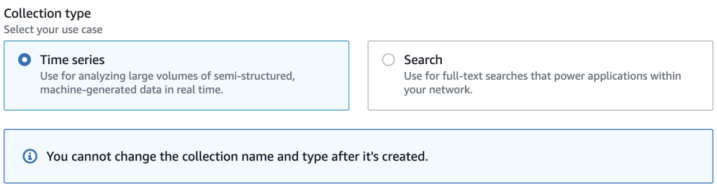
Qual è la soluzione più adatta dunque?

Come regola generale, scegli l'opzione **Dominio con standby** per i carichi di lavoro mission-critical. Per altri scenari, optiamo invece per l'altra opzione.

Serverless

AWS fornisce OpenSearch Serverless, una configurazione on-demand a scalabilità automatica per Amazon OpenSearch Service. OpenSearch Serverless fornisce raccolte, ovvero gruppi di indici con un caso d'uso comune. Le raccolte sono la controparte serverless dei cluster, ma non richiedono il provisioning manuale.

OpenSearch Serverless consente due diversi tipi di raccolte: **raccolte di serie temporali** e r**accolte di ricerca**.



La differenza principale è che nelle raccolte di ricerca tutti i dati vengono archiviati in uno storage *caldo* per garantire tempi di risposta alle query estremamente rapidi, mentre le

raccolte di serie temporali utilizzano una combinazione di cache hot e warm per ottimizzare i tempi di risposta alle query per i dati ad accesso più frequente.

Inoltre, è possibile indicizzare per ID personalizzato solo nelle raccolte di ricerca.

La capacità di calcolo OpenSearch Serveless è misurata in OpenSearch Compute Unit (OCU). Ogni OCU è una combinazione di 6 GiB di memoria, CPU virtuale corrispondente e trasferimento dati su Amazon S3.

La prima raccolta Serverless crea un'istanza di un totale di quattro OCU (2 utilizzati per l'acquisizione, primario e standby; 2 utilizzati per la ricerca, una replica attiva per l'alta disponibilità). Le raccolte successive possono condividere queste OCU. OpenSearch ridimensiona le OCU in base ai carichi di lavoro di indicizzazione e ricerca. Queste quattro OCU iniziali sono sempre attive, anche in assenza di attività di indicizzazione o ricerca, pertanto vengono addebitate almeno quattro OCU. Si può comunque impostare un numero massimo di OCU per contenere i costi.

In aggiunta, viene anche addebitato lo spazio di archiviazione conservato in Amazon S3.

Tipi di query

Ok, ora sappiamo tutto sulla creazione di un cluster OpenSearch, ma come possiamo usarlo? Attraverso le **query**.

OpenSearch fornisce un linguaggio di ricerca chiamato DSL (Query Domain-Specific Language) che fornisce un'interfaccia JSON. OpenSearch supporta anche SQL per scrivere query anziché utilizzare DSL.

Esistono due tipi principali di query: **Leaf queries** e **Compund queries**

Leaf Queries

Le leaf query ricercano un valore specificato in uno o più campi. Le leaf query possono essere ulteriormente classificate in diversi sottotipi.

Full-text queries

Le query full-text vengono utilizzate per documenti di testo. Esistono diversi tipi di query full-text.

È possibile abbinare un valore specifico in un campo specifico o cercare i valori in un elenco di campi. Si può assegnare un peso a ciascun campo per aumentarne il valore nel punteggio del risultato. Ad esempio, durante la ricerca di un libro, trovare il valore nel

campo "titolo" potrebbe avere un impatto maggiore rispetto a trovarlo nel campo "abstract".

Un tipo di query consente di cercare termini diversi in un campo e l'ultimo termine della stringa di input verrà utilizzato come prefisso per avere documenti che contengono uno qualsiasi di questi termini o termini che iniziano con il prefisso.

Alcune query full-text supportano un parametro “fuzziness” utile nel caso di input scritti con alcuni caratteri mancanti o caratteri la cui posizione viene scambiata.

Altri supportano un parametro "slop" che controlla come le parole possono essere ordinate in modo errato ed essere comunque considerate una corrispondenza.

```
GET _search
{
  "query": {
    "multi_match": {
      "query": "proud",
      "fields": ["title^2", "body"]
    }
  }
}
```

Term-level queries

Le Term-level queries vengono utilizzate per cercare nei documenti un termine specifico.

È possibile cercare un termine esatto o più termini in un campo. Durante la ricerca di più termini, le Term-level queries consentono anche di specificare il numero minimo di corrispondenze richieste. È possibile cercare un ID, un valore contenuto in un intervallo o termini che contengono un prefisso specifico.

Le query *fuzzy* cercano termini simili al termine di ricerca utilizzando la distanza di Levenshtein. Le Term-level queries offrono la possibilità di effettuare ricerche utilizzando espressioni regolari Wildcard o Lucene. Puoi anche cercare documenti che contengono un campo specifico invece di un valore.

```
GET blog/_search
{
  "query": {
    "wildcard": {
      "title": {
        "value": "*cloud"
      }
    }
  }
}
```

XY queries

Le query XY ricercano documenti che contengono geometrie utilizzando i campi xy_point e xy_shape. I campi xy_point supportano i punti. I campi xy_shape supportano punti, linee, cerchi e poligoni.

È possibile cercare documenti i cui punti o forme intersecano la forma fornita oppure documenti le cui forme non si intersecano, sono contenute o contengono la forma fornita.

```
GET index/_search
{
  "query": {
    "xy_shape": {
      "geometry": {
        "shape": {
          "type": "circle",
          "coordinates": [1.0, 5.0],
          "radius": 3
        },
        "relation": "INTERSECTS"
      }
    }
  }
}
```

Geographic queries

Le query geografiche vengono utilizzate per cercare documenti contenenti geometrie geospaziali. Le query geografiche supportano punti e forme come linee, cerchi e poligoni.

È possibile cercare documenti i cui valori geopoint si trovano all'interno di un riquadro di delimitazione o di un poligono. Le query geografiche restituiscono documenti che contengono geopoint o geoshape che intersecano la forma fornita, oppure documenti che contengono geoshape che non si intersecano, sono contenuti o contengono la forma fornita. È inoltre possibile eseguire query per documenti con punti geografici che si trovano entro una distanza specificata da un punto geografico fornito.

```
GET index/_search
{
  "query": {
    "bool": {
      "must": {
        "match_all": {}
      },
      "filter": {
        "geo_distance": {
          "distance": "10km",
          "pin.location": {
            "lat": 10,
            "lon": 50
          }
        }
      }
    }
  }
}
```

Joining queries

Le Joining queries possono essere utilizzate per interrogare un oggetto nidificato come documento indipendente o per recuperare documenti padre o documenti figlio collegati tramite un campo di tipo "join".

```
GET /_search
{
  "query": {
    "has_child": {
      "type": "child",
      "query": {
        "match_all": {}
      }
    }
  }
}
```

Compound queries

Le compound query eseguono il wrapping di più leaf query per combinarne i risultati o per modificarne il comportamento. Puoi unire più clausole di query con la logica booleana o assegnare un punteggio più alto ai documenti che corrispondono a più clausole. Le query composte offrono anche la possibilità di creare una funzione per ricalcolare il punteggio dei documenti restituiti. Infine, è possibile combinare una query “positiva” con una query “negativa”; i documenti trovati attraverso la query positiva avranno un punteggio maggiore, mentre i documenti trovati attraverso la query negativa avranno un punteggio diminuito. Questo è utile in caso di sinonimi che desideri rimuovere dai risultati.

Casi d’uso

Esistono due scopi principali per l'utilizzo del servizio OpenSearch: uno è archiviare e interrogare/analizzare i **log** e le **metriche** dell'applicazione e dell'infrastruttura, l'altro è creare un **motore di ricerca**.

Logs & Metriche

Questo è uno degli usi principali di OpenSearch. Grazie al rapido tempo di risposta alle query, consente la ricerca molto veloce di specifici log dell'applicazione o dell'infrastruttura.

Lo scopo principale di questa soluzione è disporre di un archivio di log centralizzato in cui è possibile trovare tutti i log delle applicazioni e dell'infrastruttura e creare dashboard quasi in tempo reale che visualizzano metriche diverse per comprendere l'integrità dell'applicazione.

Immaginiamo una web app composta da un front-end Single Page Application e un back-end ospitato su AWS utilizzando servizi serverless come Amazon API Gateway e AWS Lambda. Quello che si può fare è inviare tutti i log ad AWS OpenSearch strutturando un payload di log con almeno queste informazioni:

- ID richiesta: un identificatore univoco per la richiesta effettuata dal front-end.
- Tipo di registro: il tipo di registro come INFO, ERROR, DEBUG, WARNING.
- Messaggio: il tuo messaggio di log
- Timestamp: il timestamp del registro

Basterà modificare gli attributi in base a ciò che stai registrando; Ad esempio, possiamo immaginare di avere almeno due log per ogni richiesta, uno all'avvio della richiesta - che aggiungerà l'origine dell'evento come attributi aggiuntivi - e uno al termine della richiesta, con un attributo che definisce il codice di stato della risposta. Se si verifica un errore, possiamo immaginare di registrare lo stesso payload con un codice di stato di errore.

Con queste poche informazioni possiamo costruire una semplice Dashboard che ci mostri:

- Il numero di richieste al secondo
- Il numero di richieste riuscite
- Il numero di errori con il messaggio di errore associato
- Il tempo di risposta medio approssimativo dell'applicazione

e possiamo filtrare tutti i log per un intervallo di tempo specifico o anche per l'id di una singola richiesta ottenendo l'intero log di quella specifica richiesta.

Motore di ricerca

Un altro perfetto caso d'uso per OpenSearch è la creazione di un motore di ricerca. Grazie a tutti i diversi tipi di query supportati, è semplice creare una barra di ricerca che utilizza la *fuzziness* per cercare tutti i dati che assomigliano a ciò che l'utente ha digitato.

Puoi persino utilizzare la funzione *match_phrase_prefix* per completare automaticamente l'input dell'utente in base ai dati che hai nel database o utilizzare la funzione di suggerimento per correggere l'input dell'utente.

Conclusioni

In questo articolo abbiamo dato una visione generale di cos'è Opensearch, dei suoi fondamenti e di come sfruttare i servizi AWS per gestire un cluster di questo tipo.

Molti argomenti non sono stati presi in considerazione, ad esempio come autenticarsi e autorizzarsi ad un cluster OpenSearch, come sfruttare S3 per lo storage di dati warm e cold, abilitare Cognito come Identity Provider o come gestire il Disaster Recovery per le applicazioni business critical.

Se siete interessati a questi o ad altri aspetti riguardnti l'utilizzo di OpenSearch, scriveteci nei commenti!

A presto su Proud2beCloud con un nuovo blog post!

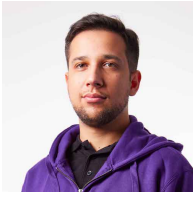
Ti è piaciuto l'articolo? Scegli gli argomenti che più ti interessano e iscriviti alla newsletter!

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!

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!)



Daniele Papa

DevOps Engineer e backend developer @ beSharp. Nel tempo libero amo giocare a videogiochi e giochi da tavolo. Negli ultimi anni mi sono avvicinato al mondo Cloud, ora passo da ruoli IAM... e giochi di ruolo.
