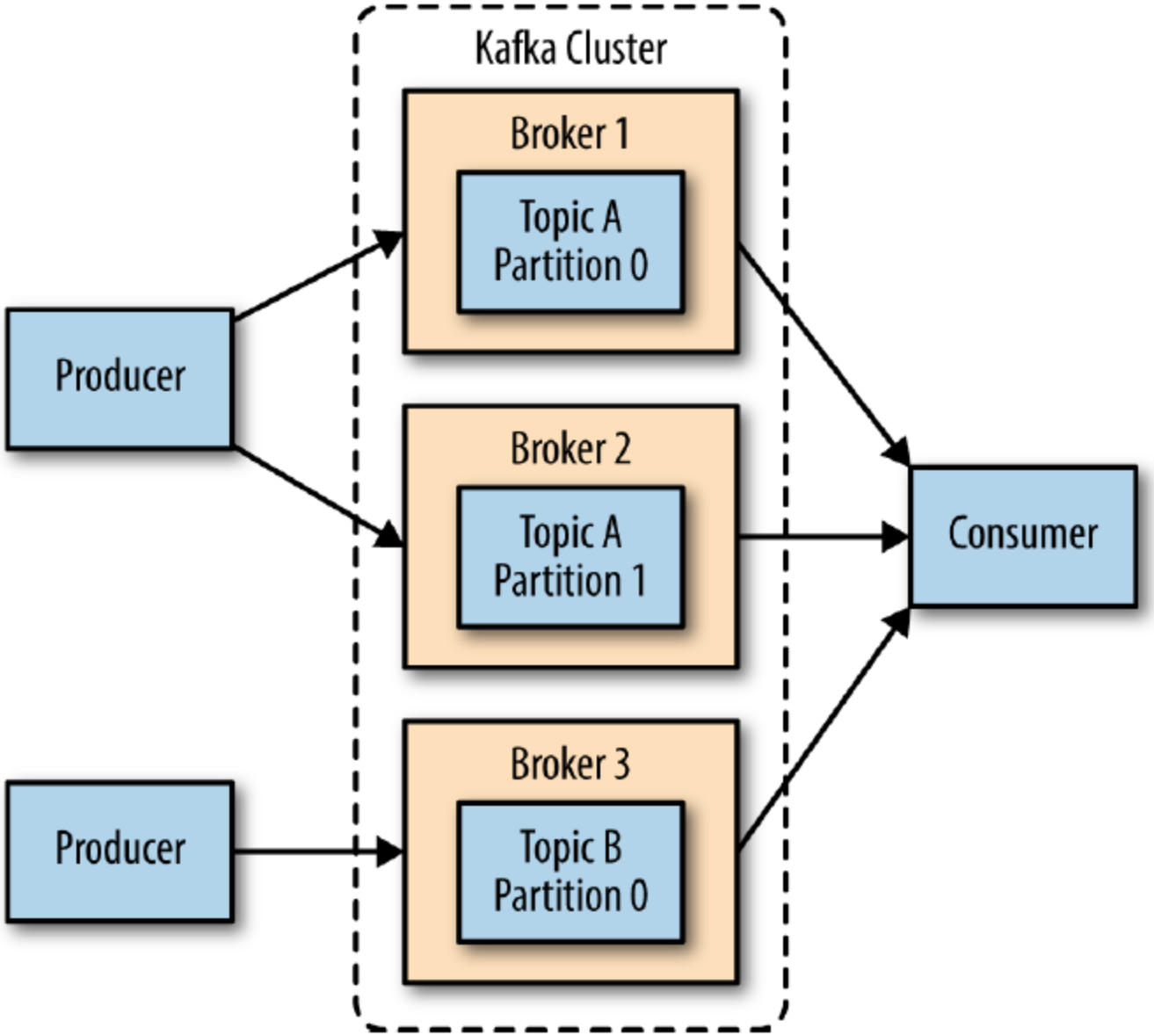

APACHE
kafka®

UNIT

Configurazione di un Broker

Un Cluster Kafka Semplice



Broker Configuration

- La configurazione di esempio fornita con la distribuzione Kafka è sufficiente per eseguire un server autonomo come POC (Proof Of Concept), ma non sarà sufficiente per la maggior parte delle installazioni.

- Esistono numerose opzioni di configurazione per Kafka che controllano tutti gli aspetti di installazione e messa a punto.

- Molte opzioni possono essere lasciate con i valori di «default», poiché trattano aspetti di ottimizzazione del broker Kafka che non saranno applicabili fino a quando non si avrà un caso d'uso specifico con cui lavorare e che richieda una regolazione/taratura (a maggior ragione se trattasi di ambienti Kafka multi-cluster).

Broker Configuration properties: broker.id

- Ogni broker Kafka deve avere un identificatore intero, impostato utilizzando la configurazione broker.id il cui valore predefinito è 0 (zero)

- La cosa più importante è che l'ID deve essere univoco all'interno di un singolo cluster Kafka.

- L'ID è arbitrario e può essere «variato tra broker» se necessario per le attività di manutenzione.

- Una buona linea guida è impostare questo valore su qualcosa di intrinseco all'host in modo che quando si esegue la manutenzione non sia oneroso mappare i numeri ID del broker sui vari host.

Broker Configuration properties: port (Deprecated)

- Il file di configurazione di esempio avvia Kafka con un listener sulla porta TCP 9092.
- Questo può essere impostato su qualsiasi porta disponibile modificando il parametro di configurazione della porta.
- Se viene scelta una porta inferiore a 1024, Kafka deve essere avviato come root.
- L'esecuzione di Kafka come root non è una configurazione consigliata.

Broker Configuration properties: zookeeper.connect

- La posizione di Zookeeper utilizzata per la memorizzazione dei metadati del broker viene impostata utilizzando il parametro di configurazione zookeeper.connect.

- Di default Kafka utilizza uno Zookeeper in esecuzione sulla porta 2181 sull'host locale (localhost: 2181). Il formato per questo parametro è un elenco separato da punti e virgola di stringhe nome host:port/path, che includono:

  - nome host: nome host o indirizzo IP del server Zookeeper.
  - port: il numero di porta del client per il server.
  - /path: un percorso Zookeeper opzionale da utilizzare come ambiente chroot per il cluster Kafka
    - Se viene omesso, viene utilizzato il percorso radice.
    - Se viene specificato un percorso chroot che non esiste, questi verrà creato dal broker all'avvio.

Perché usare un percorso chroot

- È generalmente considerata una buona pratica utilizzare un percorso chroot per il cluster Kafka.
- Ciò consente all'ensemble Zookeeper di essere condiviso con altre applicazioni, inclusi altri cluster Kafka, senza conflitti.
- È anche meglio specificare più server Zookeeper (che fanno tutti parte dello stesso insieme) in questa configurazione.
- Ciò consente al broker Kafka di connettersi a un altro membro dell'ensemble Zookeeper in caso di errore del server.

Broker Configuration properties: log.dirs

- Kafka persiste tutti i messaggi sul disco e questi segmenti di registro sono memorizzati nelle directory specificate nella configurazione di log.dirs.

- Questo è un elenco separato da virgole di percorsi sul sistema locale.

- Se viene specificato più di un percorso, il broker memorizzerà le partizioni su di esse in modalità «least-used» (meno utilizzato) con i segmenti di registro di una partizione memorizzati nello stesso percorso.

- Si noti che il broker inserirà una nuova partizione nel percorso che contiene il minor numero di partizioni attualmente memorizzate al suo interno e non la minima quantità di spazio su disco utilizzata quando siamo negli scenari descritti dalle seguenti slide.

Broker Configuration properties: log.dirs. -> num.recovery.threads.per.data.dir

- Kafka utilizza un pool configurabile di thread per la gestione dei segmenti di registro. Attualmente, viene utilizzato questo pool di thread:
  - Quando si avvia normalmente, per aprire i segmenti di registro di ciascuna partizione
  - Quando si avvia dopo un errore, per controllare e troncare i segmenti di registro di ciascuna partizione
  - Durante lo spegnimento, per chiudere in modo pulito i segmenti di registro
- Per impostazione predefinita, viene utilizzato solo un thread per directory di registro.

Broker Configuration properties: log.dirs. -> num.recovery.threads.per.data.dir

- Poiché tali thread vengono utilizzati solo durante l'avvio e l'arresto, è ragionevole impostare un numero maggiore di thread per parallelizzare le operazioni.

- In particolare, quando si esegue il ripristino da un arresto improprio, ciò può comportare la differenza di diverse ore quando si riavvia un broker con un numero elevato di partizioni!

- Quando si imposta questo parametro, ricordare che il numero configurato è per directory di registro specificata con log.dirs.

- Ciò significa che se num.recovery.threads.per.data.dir è impostato su 8 e ci sono 3 percorsi specificati in log.dirs, si tratta di un totale di 24 thread.

Broker Configuration properties: log.dirs. -> auto.create.topics.enable

- La configurazione predefinita di Kafka specifica che il broker può creare automaticamente un argomento nelle seguenti circostanze:
  - Quando un producer inizia a scrivere messaggi sull'argomento
  - Quando un consumer inizia a leggere i messaggi dall'argomento
  - Quando un client richiede metadati per l'argomento
- In molte situazioni, questo può essere un comportamento indesiderato, soprattutto perché non esiste alcun modo per convalidare l'esistenza di un argomento attraverso il protocollo Kafka senza far sì che venga creato.
- Se si sta gestendo esplicitamente la creazione di argomenti, manualmente o tramite un sistema di provisioning, è possibile impostare la configurazione auto.create.topics.enable su false.

UNIT

Topic Defaults

Topic: num.partitions

- Il parametro num.partitions determina il numero di partizioni con cui viene creato un nuovo argomento, ciò quando è abilitata la creazione automatica dell'argomento (impostazione predefinita).

- Questo parametro viene settato automaticamente su una partizione; tieni presente che il numero di partizioni per un argomento può solo essere aumentato, mai diminuito.

- Ciò significa che se un argomento deve avere meno partizioni rispetto a num.partitions sarà necessario prestare attenzione per creare manualmente l'argomento.

Topic: num.partitions

- Le partizioni sono il modo in cui un argomento viene scalato all'interno di un cluster Kafka, il che rende importante utilizzare il numero delle partizioni che bilanceranno il carico dei messaggi nell'intero cluster man mano che vengono aggiunti i broker.

- Molti utenti avranno il numero delle partizioni per un argomento uguale o multiplo del numero di broker nel cluster.

- Ciò consente alle partizioni di essere distribuite uniformemente ai broker, che distribuiranno uniformemente il carico del messaggio.

- Questo non è un requisito, tuttavia, poiché è anche possibile bilanciare il carico dei messaggi con più argomenti.

Come scegliere il numero di partizioni

- Esistono diversi fattori da considerare nella scelta del numero di partizioni.

- Qual è il rendimento atteso per l'argomento? Ad esempio, ci si aspetta di scrivere 100 KB al secondo o 1 GB al secondo?

- Qual è la velocità massima che ci si aspetta di ottenere dal consumo di una singola partizione? Al massimo si avrà sempre un consumatore che legge da una partizione, quindi nel caso in cui il consumer più lento che scrive i dati in un database e questo database non gestisce mai più di 50 MB al secondo da ogni thread che scrive su di esso, allora si può desumere che si è limitati a un throughput di 60 MB quando viene consumato da una partizione.

Come scegliere il numero di partizioni

- È possibile eseguire lo stesso esercizio per stimare il throughput massimo per producer per una singola partizione, ma poiché i producer sono in genere molto più veloci dei consumer, di solito è sicuro saltarlo.

- Se si inviano messaggi alle partizioni in base alle chiavi, l'aggiunta di partizioni in un secondo momento può essere molto impegnativa, quindi è necessario calcolare il throughput in base all'utilizzo futuro previsto, non all'uso corrente.

- Considerare il numero di partizioni che verranno posizionate su ciascun broker, lo spazio su disco disponibile e la larghezza di banda della rete per broker.

- Evitare di sopravvalutare, poiché ogni partizione utilizza memoria e altre risorse sul broker ciò si tradurrà in un aumento del tempo per le negoziazione dei leader.

Come scegliere il numero di partizioni

- E’ chiaro che avere diverse partizioni è utile ma non bisogna esagerare.
- Se si dispone di una stima del throughput target dell'argomento e del throughput atteso dei consumer, è possibile dividere il throughput target per il throughput atteso del consumer e ricavare il numero di partizioni in questo modo.
- Quindi, se voglio essere in grado di scrivere e leggere 1 GB/sec da un argomento e so che ogni consumer può elaborare solo 50 MB/s, allora so che ho bisogno di almeno 20 partizioni.
- In questo modo, posso avere 20 consumer che leggono dall'argomento e raggiungere 1 GB/sec.
- Se non si dispone di queste informazioni dettagliate, l’esperienza suggerisce che limitare le dimensioni della partizione sul disco a meno di 6 GB al giorno di conservazione spesso dà risultati soddisfacenti.

Topic: log.retention.ms

- La durata massima del tempo in cui Kafka conserverà i messaggi è determinata dal parametro log.retention.hours la cui impostazione predefinita è 168 ore (o una settimana).

- Tuttavia, vi sono altri due parametri consentiti, log.retention.minutes e log.retention.ms.

- Tutti e tre specificano la stessa configurazione, ovvero la quantità di tempo dopo la quale è possibile eliminare i messaggi, ma il parametro consigliato da utilizzare è log.retention.ms, poiché la dimensione dell'unità più piccola avrà la precedenza se ne viene specificata più di una volta.

- Questo farà in modo che il valore impostato per log.retention.ms sia sempre quello utilizzato.

- Se viene specificato più di un parametro, la dimensione dell'unità più piccola avrà la precedenza.

Topic: Retention By Time e Last Modified Times

- La «Retention by time» viene eseguita esaminando l'ora dell'ultima modifica (mtime) su ciascun file del segmento di log sul disco.

- In normali operazioni del cluster, questo è il momento in cui il segmento di log è stato chiuso e rappresenta la data/ora dell'ultimo messaggio nel file.

- Quando si utilizzano gli strumenti di amministrazione per spostare le partizioni tra i broker, il processo non è ottimizzato e comporterà una «retention» eccessiva.

Topic: log.retention.bytes

- Un altro modo per far scadere i messaggi si basa sul numero totale di byte dei messaggi conservati.
- Questo valore viene settato utilizzando il parametro log.retention.bytes e viene applicato per partizione.
- Ciò significa che se si dispone di un argomento con 8 partizioni e log.retention.bytes è impostato su 1 GB, la quantità di dati conservati per l'argomento sarà al massimo di 8 GB.
- Si noti che tutta la «retention» viene eseguita per singole partizioni, non per l'argomento.
- Ciò significa che se il numero di partizioni per un argomento viene espanso, la «retention» aumenterà anche se viene utilizzato log.retention.bytes.

- Se è stato specificato un valore per log.retention.bytes e log.retention.ms (o un altro parametro per la «retention» nel tempo), i messaggi possono essere rimossi quando viene soddisfatto uno dei due criteri.

- Ad esempio, se log.retention.ms è impostato su 86.400.000 (1 giorno) e log.retention.bytes è impostato su 1.000.000.000 (1 GB), è possibile che:
  - vengano eliminati i messaggi che hanno meno di 1 giorno se il totale il volume dei messaggi nel corso della giornata è superiore a 1 GB.
  - al contrario, se il volume è inferiore a 1 GB, i messaggi possono essere eliminati dopo 1 giorno anche se la dimensione totale della partizione è inferiore a 1 GB.

Topic: log.segment.bytes

- I settings del «log-retention» precedentemente menzionati operano sui «log segment», non sui singoli messaggi.
- Man mano che i messaggi vengono prodotti sul broker Kafka, vengono aggiunti al «log segment» corrente per la partizione.
- Una volta che il «log segment» ha raggiunto la dimensione specificata dal parametro log.segment.bytes, il cui valore predefinito è 1 GB, il «log segment» viene chiuso e ne viene aperto uno nuovo.
- Una volta che un «log segment» è stato chiuso viene considerato scaduto.
- Una dimensione inferiore del «log segment» indica che i file devono essere chiusi e allocati più spesso, riducendo l'efficienza complessiva delle scritture su disco.

Topic: log.segment.bytes

- La regolazione delle dimensioni dei «log segment» può essere importante se gli argomenti hanno un basso tasso di produzione.

- Ad esempio, se un argomento riceve solo 100 megabyte al giorno di messaggi e log.segment.bytes è impostato sul valore predefinito, saranno necessari 10 giorni per riempire un segmento.

- Poiché i messaggi non possono scadere fino alla chiusura del segmento di registro, se log.retention.ms è impostato su 604.800.000 (1 settimana), in realtà saranno conservati fino a 17 giorni di messaggi e ciò fino alla scadenza del «log segment» chiuso.

- Questo perché una volta chiuso il «log segment» con gli attuali 10 giorni di messaggi, questi deve essere conservato per 7 giorni prima che scada in base alla politica temporale (poiché il segmento non può essere rimosso fino a quando l'ultimo messaggio nel segmento scada).

Topic: Retrieving Offsets by Timestamp

- La dimensione del «log segment» influenza anche il comportamento del recupero degli offset in base al timestamp.
- Quando si richiedono offset per una partizione in un momento specifico, Kafka trova il file del «log segment» che era stato scritto in quel momento.
- Lo fa utilizzando la creazione e l'ora dell'ultima modifica del file e cercando un file che è stato creato prima del timestamp specificato e modificato l'ultima volta dopo il timestamp.
- L'offset all'inizio di quel «log segment» (che è anche il nome del file) viene restituito nella risposta.

Topic: log.segment.ms

- Un altro modo per controllare quando i «log segment» sono chiusi è utilizzare il parametro log.segment.ms, che specifica la quantità di tempo dopo la quale un «log segment» deve essere chiuso.

- Come per i parametri log.retention.bytes e log.retention.ms, log.segment.bytes e log.segment.ms non sono proprietà reciprocamente esclusive.

- Kafka chiuderà un «log segment» quando viene raggiunto il limite di dimensioni o quando viene raggiunto il limite di tempo, a seconda dell'evento che si verifica per primo.

- Per impostazione predefinita, non esiste alcuna impostazione per log.segment.ms, che comporta la chiusura dei «log segment» solo per dimensione.

Topic: Disk Performance quando si usa il Time-Based Segments

- Quando si utilizza un limite per i «log segment» basato sul tempo, è importante considerare l'impatto sulle prestazioni del disco quando più «log segment» vengono chiusi contemporaneamente.

- Questo può accadere quando vi sono molte partizioni che non raggiungono mai il limite di dimensione per i «log segment» , poiché l'orologio per il limite di tempo inizierà all'avvio del broker e verrà eseguito sempre allo stesso tempo per queste partizioni a basso volume.

Topic: message.max.bytes

- Il broker Kafka limita la dimensione massima di un messaggio che può essere prodotto, configurato con il parametro message.max.bytes, il cui valore predefinito è 1.000.000 o 1 MB.

- Un producer che tenta di inviare un messaggio più grande di questo riceverà un errore dal broker e il messaggio non verrà accettato.

- Come per tutte le dimensioni di byte specificate sul broker, questa configurazione si occupa della dimensione dei messaggi compressi, il che significa che i producer possano inviare messaggi molto più grandi di questo valore non compresso, a condizione che vengano compressi nella dimensione message.max.bytes configurata.

Topic: message.max.bytes

- Vi sono notevoli effetti sulle prestazioni derivanti dall'aumento delle dimensioni consentite dei messaggi.

- Messaggi più grandi implica che i thread del broker (che si occupano dell'elaborazione delle connessioni di rete e delle richieste) lavoreranno più a lungo su ogni richiesta.

- I messaggi più grandi aumentano anche le dimensioni delle scritture su disco, con conseguenze sulla velocità di I / O.

Topic: Coordinamento delle configurazioni delle dimensioni dei messaggi

- La dimensione del messaggio configurata sul broker Kafka deve essere coordinata con la configurazione fetch.message.max.bytes sui client consumer.

- Se questo valore è inferiore a message.max.bytes, i consumer che incontrano messaggi più grandi non riusciranno a recuperare quei messaggi, causando una situazione in cui il consumer si blocca e non può procedere.

- La stessa regola si applica alla configurazione replica.fetch.max.bytes sui broker quando configurata in un cluster.

UNIT

Hardware Selection

Hardware Selection

- La selezione di una configurazione hardware appropriata per un broker Kafka può essere considerata più un'arte che una scienza.

- Kafka stesso non ha requisiti rigorosi su una specifica configurazione hardware e funzionerà senza problemi su qualsiasi sistema.

- Una volta che le prestazioni diventano un problema, tuttavia, ci sono diversi fattori che contribuiranno alle prestazioni complessive: throughput e capacità del disco, memoria, rete e CPU.

- Dopo aver determinato quali tipi di prestazioni sono i più critici per l'ambiente, si è in grado di selezionare una configurazione hardware ottimizzata adatta al budget a disposizione (ovviamente si parla di dimensionamento dell'Hardware).

Disk Throughput

- Le prestazioni dei producer client saranno influenzate direttamente dal throughput del disco del broker utilizzato per l'archiviazione dei «log-segment».

- I messaggi di Kafka devono essere assegnati all'archiviazione locale quando vengono prodotti e la maggior parte dei client attende fino a quando almeno un broker avrà confermato che i messaggi sono stati impegnati prima di considerare l'invio riuscito.

- Ciò significa che scritture su disco più veloci equivalgono a una latenza di produzione inferiore.

Disk Throughput

- La decisione ovvia quando si tratta di throughput del disco è se utilizzare dischi rigidi tradizionali (HDD) o dischi allo stato solido (SSD).
- Gli SSD hanno tempi di ricerca e accesso drasticamente più bassi e forniranno le migliori prestazioni.
- D'altro canto, gli HDD sono più economici e offrono una maggiore capacità per unità.
- È inoltre possibile migliorare le prestazioni degli HDD utilizzandone diversi per un broker, sia con più directory di dati che impostando le unità in una configurazione «redundant array of independent disks» (RAID).
- Altri fattori, come la tecnologia specifica dell'unità (ad es. Memoria collegata in serie o ATA seriale), nonché la qualità del controller dell'unità, influiranno sulla velocità effettiva.

Disk Capacity

- La capacità è l'altra volto della discussione sull'archiviazione.
- La quantità di capacità del disco necessaria è determinata dalla modalità di conservazione dei messaggi in qualsiasi momento.
- Se si prevede che il broker riceva 1 TB di traffico ogni giorno, con 7 giorni di conservazione, il broker avrà bisogno di un minimo di 7 TB di spazio di archiviazione utilizzabile per i «log-segment».
- Bisogna inoltre considerare un overhead di almeno il 10% per altri file, oltre a qualsiasi buffer che si desideri mantenere per variazioni del traffico o crescita nel tempo.

Disk Capacity

- La capacità di archiviazione è uno dei fattori da considerare quando si dimensiona un cluster Kafka e si determina quando espanderlo.

- Il traffico totale per un cluster può essere bilanciato su di esso con più partizioni per argomento, il che consentirà ai broker aggiuntivi di aumentare la capacità disponibile se la densità su un singolo broker non sarà sufficiente.

- La decisione sulla quantità di capacità del disco necessaria dovrà prendere in considerazione anche la strategia di replica scelta per il cluster.

Memory

- Il normale modo di operare per un consumer di Kafka sta leggendo dalla fine delle partizioni, dove il consumatore viene catturato e in ritardo rispetto ai produttori molto poco, se non del tutto.

- In questa situazione, i messaggi che il consumatore sta leggendo vengono archiviati in modo ottimale nella cache della pagina del sistema, con conseguenti letture più rapide rispetto al caso in cui il broker debba rileggere i messaggi dal disco.

- Pertanto, avere più memoria disponibile nel sistema per la cache delle pagine migliorerà le prestazioni dei client consumer.

Memory

- Kafka non ha bisogno di molta memoria heap configurata per la Java Virtual Machine (JVM).
- Anche un broker che gestisce X messaggi al secondo e una velocità dati di X megabit al secondo può essere eseguito con un heap da 5 GB.
- Il resto della memoria di sistema verrà utilizzato a beneficio di Kafka consentendo al sistema di memorizzare nella cache i «log-segment» in uso.
- Questo è il motivo principale per cui non è consigliabile posizionare Kafka su un sistema con qualsiasi altra applicazione significativa, poiché dovranno condividere l'uso della cache.
- Ciò ridurrà le prestazioni dei consumatori per Kafka.

- Il throughput di rete disponibile specificherà la quantità massima di traffico che Kafka può gestire.

- Questo è spesso il fattore determinante, combinato con l'archiviazione su disco, per il dimensionamento dei cluster.

- Ciò è complicato dallo squilibrio intrinseco tra l'utilizzo della rete in entrata e in uscita creato dal supporto di Kafka per più consumer.

- Un producer può scrivere 1 MB al secondo per un determinato argomento, ma potrebbe esservi un numero qualsiasi di consumer che creano un sovraccarico sull'utilizzo della rete in uscita.

- Altre operazioni come la replica dei cluster e il mirroring aumenteranno anche i requisiti hardware richiesti.

- Se l'interfaccia di rete si satura, non è raro che la replica del cluster rimanga indietro, il che può lasciare il cluster in uno stato di vulnerabilità.

CPU

- La potenza di elaborazione non è importante quanto il disco e la memoria, ma influirà in una certa misura sulle prestazioni complessive del broker.
- Idealmente, i client dovrebbero comprimere i messaggi per ottimizzare l'utilizzo della rete e del disco.
- Il broker Kafka deve decomprimere tutti i batch dei messaggi, tuttavia, al fine di convalidare il checksum dei singoli messaggi e assegnare un'offset.
- Deve quindi ricomprimere il batch di messaggi per archiviarlo su disco.
- Questi è il punto da dove derivano la maggior parte dei requisiti di Kafka per la potenza di elaborazione anche se, tuttavia, non dovrebbe essere il fattore principale nella scelta dell'hardware.

Kafka nel Cloud

- L’installazione ideale di Kafka è all'interno di ambienti di cloud computing, come Amazon Web Services (AWS). AWS offre molte istanze di calcolo, ognuna con una diversa combinazione di CPU, memoria e disco, pertanto è necessario dare priorità alle varie caratteristiche di prestazione di Kafka per selezionare la configurazione di istanza corretta da utilizzare.

- Un buon punto di partenza è la quantità di «retention» dei dati richiesta, seguita dalle prestazioni richieste dai producer.

- Se è necessaria una latenza molto bassa, potrebbero essere necessarie istanze ottimizzate di I / O con memoria SSD locale.

- Altrimenti, l'archiviazione effimera (come AWS Elastic Block Store) potrebbe essere sufficiente. Una volta prese queste decisioni, le opzioni di memoria e CPU disponibili saranno appropriate per le prestazioni.

- In termini reali, ciò significa che per AWS i tipi di istanza m4 o r3 sono una scelta comune.

- L'istanza m4 consentirà periodi di conservazione maggiori, ma il throughput sul disco sarà inferiore perché si trova nello storage a «elastic block».

- L'istanza r3 avrà un throughput molto migliore con le unità SSD locali, ma tali unità limiteranno la quantità di dati che possono essere conservati.

- Per il meglio di entrambi i mondi, è necessario passare ai tipi di istanza i2 o d2, che sono significativamente più costosi.

UNIT

Kafka Clusters

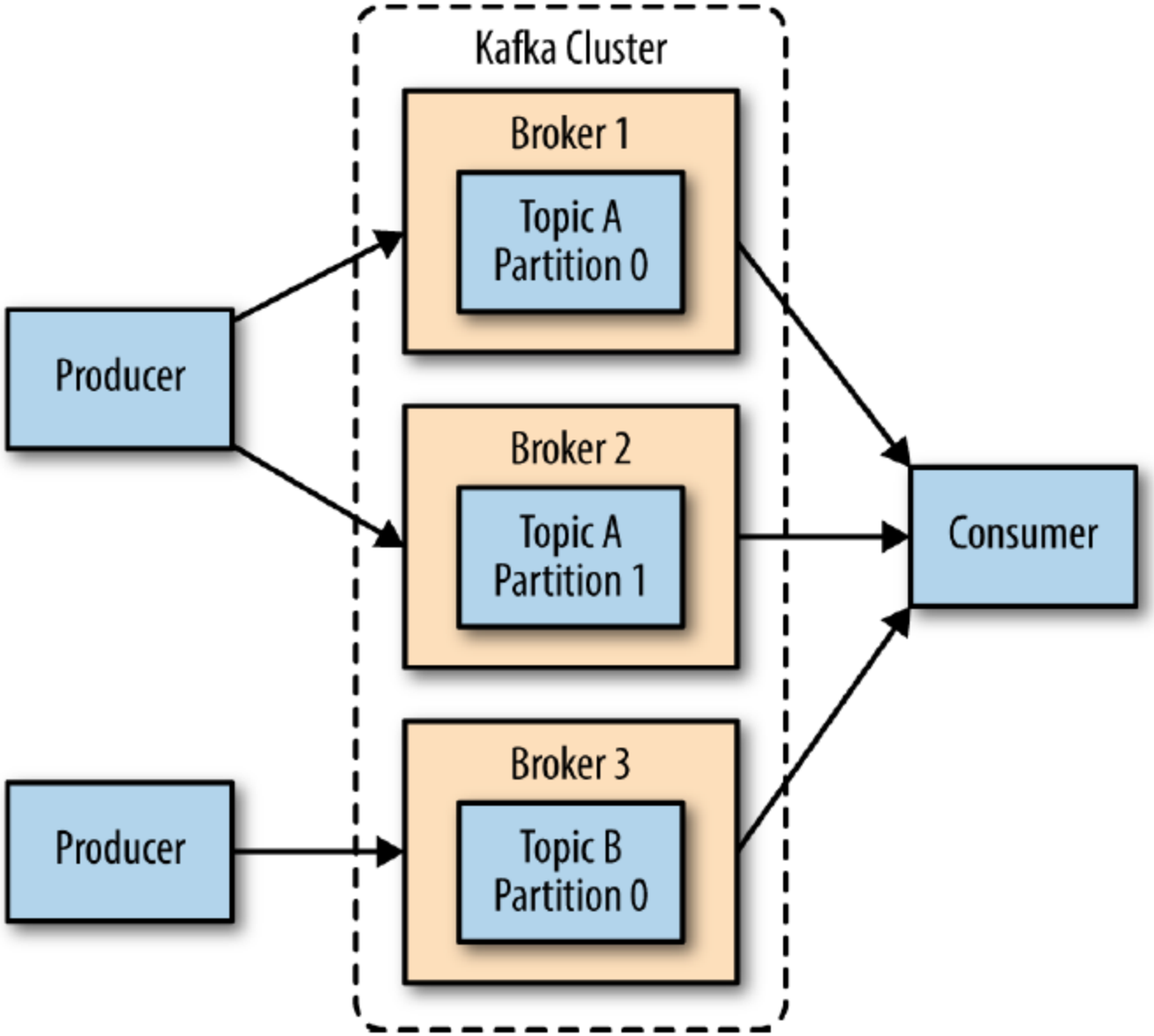
Kafka Clusters

- Un singolo server Kafka va bene per un ambiente di sviluppo locale o per un POC; in tutti gli altri scenari vi sono vantaggi significativi nell'avere più broker configurati come cluster

- Il più grande vantaggio è la capacità di ridimensionare il carico su più server. L'usare Broker che usino la stessa replica previene dalla perdita di dati dovuta a guasti di un singolo sistema.

- La replica consentirà inoltre di eseguire lavori di manutenzione su Kafka o sui sistemi sottostanti mantenendo comunque la disponibilità per i clienti.

Kafka Clusters



Quanti broker è necessario configurare?

- La dimensione appropriata per un cluster Kafka è determinata da diversi fattori.
- Il primo fattore da considerare è la quantità di capacità del disco necessaria per conservare i messaggi e la quantità di memoria disponibile su un singolo broker.
- Se il cluster deve conservare 10 TB di dati e un singolo broker può archiviare 2 TB, la dimensione minima del cluster è di cinque broker.
- Inoltre, l'utilizzo della replica aumenterà i requisiti di archiviazione di almeno il 100%, a seconda del fattore di replica scelto.
- Ciò significa che questo stesso cluster, configurato con la replica, ora deve contenere almeno 10 broker.

Quanti broker è necessario configurare?

- L'altro fattore da considerare è la capacità del cluster di gestire le richieste.

- Le domande per le quali trovare le risposte sono, ad esempio:
  - Qual è la capacità delle interfacce di rete ?
  - Queste ultime possono gestire il traffico client se vi sono più consumer concorrenti o se il traffico non è coerente durante il periodo di «retention» dei dati (ad esempio, picchi di traffico durante le ore di punta) ?

- Se l'interfaccia di rete su un singolo broker viene utilizzata all'80% della capacità al massimo e ci sono due consumer di tali dati, i consumer stessi non saranno in grado di tenere il passo con il picco del traffico se non ci sono due broker.

Quanti broker è necessario configurare?

- Se la replica viene utilizzata nel cluster, si tratta di un ulteriore consumer dei dati che devono essere presi in considerazione.

- Si potrebbe anche pensare di scalare su più broker in un cluster al fine di gestire i problemi di prestazioni causati dalla minore velocità effettiva del disco o dalla memoria di sistema disponibile.

Broker Configuration

- Esistono solo due requisiti nella configurazione del broker per consentire a più broker Kafka di unirsi a un singolo cluster.
- Il primo è che tutti i broker devono avere la stessa configurazione per il parametro zookeeper.connect. Questo specifica l'ensemble Zookeeper e il percorso in cui il cluster memorizza i metadati.
- Il secondo requisito è che tutti i broker nel cluster devono avere un valore univoco per il parametro broker.id.
- Se due broker tentano di unirsi allo stesso cluster con lo stesso broker.id, il secondo broker registrerà un errore e non si avvierà.
- Esistono diversi altri parametri di configurazione utilizzati durante l'esecuzione di un cluster.

UNIT

OS Tuning

OS Tuning

- Mentre la maggior parte delle distribuzioni Linux hanno una configurazione pronta all'uso per i parametri di «kerneltuning» che funzionano abbastanza bene per la maggior parte delle applicazioni, ci sono alcune modifiche che possono essere fatte per un broker Kafka che ne miglioreranno le prestazioni.

- Il Tuning ruota principalmente attorno alla memoria virtuale e ai sottosistemi di rete, oltre alle specifiche per il punto di «mount» del disco utilizzato per l'archiviazione dei log segment.

- Questi parametri sono in genere configurati nel file /etc/sysctl.conf, ma è necessario fare riferimento alla documentazione della propria distribuzione Linux per dettagli specifici su come regolare la configurazione del kernel.

OS Tuning: Virtual Memory

- In generale, il sistema di memoria virtuale Linux si adatterà automaticamente al carico di lavoro del sistema ma possiamo apportare alcune modifiche sia al modo in cui viene gestito lo spazio di «swap», sia la gestione delle pagine di «memoria sporche», per ottimizzare l'OS ai fini del carico di lavoro di Kafka.

- Come con la maggior parte delle applicazioni, in particolare quelle in cui la velocità effettiva è una preoccupazione, è meglio evitare lo «swapping» sul disco a (quasi) tutti i costi.

- I costi sostenuti per lo «swap» di pagine di memoria su disco sono da considerarsi come un impatto evidente su tutti gli aspetti delle prestazioni in Kafka.

- Inoltre, Kafka fa un uso intensivo della «system page cache» e, se il sistema VM sta «swappando» (lasciatemi passare il verbo ☺) su disco, non è disponibile memoria sufficiente per la «page cache».

OS Tuning: Virtual Memory

- Un modo per evitare lo «swap» è semplicemente quello di non configurare alcuno spazio di scambio. Lo «swap» non è un requisito, ma fornisce una rete di sicurezza se succede qualcosa di catastrofico sul sistema.

- Lo «swap» può impedire al sistema operativo di interrompere improvvisamente un processo a causa di una condizione di memoria insufficiente.

- Per questo motivo, si consiglia di impostare il parametro vm.swappiness su un valore molto basso, come 1.

- Il parametro è una percentuale della probabilità che il sottosistema VM utilizzi lo spazio di «swap» anziché eliminare le pagine di memoria dalla «page cache».

- È preferibile ridurre la dimensione della «page cache» piuttosto che «swappare».

OS Tuning: Virtual Memory -> Perchè non impostare lo swap a 0?

- «Un tempo» la raccomandazione per vm.swappiness era di settarlo sempre a 0.
- Questo valore aveva come significato: "non effettuare lo swap a meno che non vi sia una condizione di memoria insufficiente".
- Tuttavia, il significato di questo valore è cambiato a partire dalla versione 3.5-rc1 del kernel Linux e tale modifica è stata importata in molte distribuzioni, inclusi i kernel Red Hat Enterprise Linux a partire dalla versione 2.6.32-303.
- Ciò ha modificato il significato del valore 0 in "mai effettuare lo swap, in nessun caso".
- È per questo motivo che ora è raccomandato un valore del parametro pari a 1.

OS Tuning: Virtual Memory

- C'è anche un vantaggio nel regolare il modo in cui il kernel gestisce le pagine di memoria sporche di cui deve essere effettuato il «flush» sul disco.

- Kafka si affida alle prestazioni di I/O del disco per fornire buoni tempi di risposta ai producer.

- Questo è anche il motivo per cui i log segment vengono generalmente inseriti su un disco veloce, sia che si tratti di un singolo disco con un tempo di risposta rapido (ad es. SSD) o di un sottosistema di dischi con NVRAM significativa per la memorizzazione nella cache (ad es. RAID).

OS Tuning: Virtual Memory

- Il risultato è che il numero di pagine sporche consentite può essere ridotto prima che il processo di «flush» in background inizi a scriverle su disco.

- Ciò si ottiene settando il parametro vm.dirty_background_ratio su un valore inferiore a 10.

- Il valore è una percentuale della quantità totale di memoria di sistema e l'impostazione di questo valore su 5 è appropriata in molte situazioni.

- Questo parametro non dovrebbe mai essere settato a 0 in quanto ciò causerebbe il flush continuo delle pagine del kernel, che eliminerebbe quindi la capacità del kernel di bufferizzare le scritture su disco contro picchi temporanei nelle prestazioni del dispositivo sottostante.

OS Tuning: Virtual Memory

- Il numero totale di pagine sporche consentite prima che il kernel imponga alle operazioni sincrone di effettuarne il «flush» sul disco può anche essere aumentato modificando il valore di vm.dirty_ratio portandolo al di sopra del valore predefinito di 20 (una percentuale della memoria totale del sistema) .

- Vi è una vasta gamma di valori possibili per questa impostazione, ma tra 60 e 80 è un numero ragionevole.

- Questa impostazione presenta una piccola quantità di rischio, sia per quanto riguarda la quantità di attività del disco «unflushed», sia per il potenziale di lunghe pause di I/O in caso di forzatura dei flush sincroni.

- Se viene scelta un'impostazione più alta per vm.dirty_ratio, si consiglia vivamente di utilizzare la replica nel cluster Kafka per evitare errori di sistema.

OS Tuning: Virtual Memory

- Quando si scelgono i valori per questi parametri, è consigliabile rivedere il numero di pagine sporche nel tempo mentre il cluster Kafka è in esecuzione sotto carico (sia in produzione che simulato).

- Il numero corrente di pagine sporche può essere determinato controllando il file/proc/vmstat:

```
# cat /proc/vmstat | egrep "dirty|writeback"
nr_dirty 3875
nr_writeback 29
nr_writeback_temp 0
#
```

OS Tuning: Disk

- Oltre la selezione dei tipi di disk (nonché della configurazione del RAID se utilizzato), la scelta del filesystem da utilizzare può avere un elevato impatto sulle prestazioni.

- Sono disponibili molti tipi di file system, ma le opzioni più comuni per i file system locali sono EXT4 (quarto file system esteso) o Extents File System (XFS).

- Di recente, XFS è diventato il filesystem predefinito per molte distribuzioni Linux, e questo per una buona ragione: supera EXT4 per la maggior parte dei carichi di lavoro con l'ottimizzazione minima richiesta.

- EXT4 può funzionare bene, ma richiede l'utilizzo di parametri di ottimizzazione considerati meno sicuri.

OS Tuning: Disk

- Ciò include l'impostazione dell'intervallo di commit su un tempo più lungo rispetto al valore predefinito di 5 per forzare i flush meno frequenti.

- EXT4 ha inoltre introdotto l'allocazione ritardata dei blocchi, il che comporta maggiori possibilità di perdita di dati e danneggiamento del file system in caso di guasto del sistema.

- Il filesystem XFS utilizza anche un algoritmo di allocazione ritardata, ma è generalmente più sicuro di quello utilizzato da EXT4. XFS offre anche prestazioni migliori per il carico di lavoro di Kafka senza richiedere ottimizzazione (oltre all'ottimizzazione automatica eseguita dal filesystem).

- È anche più efficiente quando si scrivono in batch le scritture su disco, che si combinano per fornire una migliore velocità di I/O complessiva.

OS Tuning: Disk

- Indipendentemente dal file system scelto per il mount che contiene i log segment, si consiglia di settare l'opzione di montaggio «noatime» per il punto di mount.

- I metadati del file contengono tre timestamp:

  - ora di creazione (ctime)
  - ora dell'ultima modifica (mtime)
  - ora dell'ultimo accesso (atime).

- Per impostazione predefinita, l'atime viene aggiornato ogni volta che viene letto un file. Ciò genera un gran numero di scritture su disco.

OS Tuning: Disk

- L'attributo atime è generalmente considerato di scarsa utilità, a meno che un'applicazione non debba sapere se è stato effettuato l'accesso a un file dall'ultima modifica (nel qual caso è possibile utilizzare l'opzione in tempo reale).

- L'atime non è affatto usato da Kafka, quindi disabilitarlo è sicuro da fare.

- L'impostazione di noatime sul mount impedirà il verificarsi di questi aggiornamenti di data e ora, ma non influirà sulla corretta gestione degli attributi ctime e mtime.

- La regolazione dell'ottimizzazione predefinita dello stack di rete Linux è comune per qualsiasi applicazione che generi un'elevata quantità di traffico di rete, poiché il kernel non è ottimizzato (per impostazione predefinita) per trasferimenti di dati di grandi dimensioni e ad alta velocità.

- In effetti, le modifiche consigliate per Kafka sono le stesse suggerite per la maggior parte dei server Web e altre applicazioni di rete.

- La prima regolazione consiste nel modificare la quantità massima e predefinita di memoria allocata per i buffer di invio e ricezione per ciascun socket.

- Ciò aumenterà significativamente le prestazioni per trasferimenti di grandi dimensioni.

- I parametri pertinenti le dimensioni predefinite del buffer di invio e ricezione per socket sono net.core.wmem_default e net.core.rmem_default e un'impostazione ragionevole per questi parametri è 131.072 o 128 KiB.

- I parametri per le dimensioni massime del buffer di invio e ricezione sono net.core.wmem_max e net.core.rmem_max e un'impostazione ragionevole è 2.097.152 o 2 MiB.

- Bisogna tener presente che la dimensione massima non indica che a ogni socket verrà assegnato questo spazio di buffer; permette solo fino a quel punto, se necessario.

- Oltre alle impostazioni del socket, le dimensioni del buffer di invio e ricezione per i socket TCP devono essere impostate separatamente utilizzando i parametri net.ipv4.tcp_wmem e net.ipv4.tcp_rmem.

- Questi vengono impostati utilizzando tre numeri interi separati da spazio che specificano rispettivamente le dimensioni minima, predefinita e massima.

- La dimensione massima non può essere superiore ai valori specificati per tutti i socket utilizzando net.core.wmem_max e net.core.rmem_max.

- Un'impostazione di esempio per ciascuno di questi parametri è "4096 65536 2048000", che è un minimo di 4 KiB, un valore predefinito di 64 KiB e un buffer massimo di 2 MiB.

- In base al carico di lavoro effettivo dei broker Kafka, potresti voler aumentare le dimensioni massime per consentire un maggiore buffering delle connessioni di rete.

- Esistono diversi altri parametri di ottimizzazione della rete che sono utili da settare.

- L'abilitazione del ridimensionamento della finestra TCP impostando net.ipv4.tcp_window_scaling su 1 consentirà ai client di trasferire i dati in modo più efficiente e consentirà il buffering dei dati sul lato broker.

- L'aumento del valore di net.ipv4.tcp_max_syn_backlog al di sopra del valore predefinito di 1024 consentirà di accettare un numero maggiore di connessioni simultanee.

- L'aumento del valore di net.core.netdev_max_backlog a un valore superiore a quello predefinito di 1000 può aiutare a far aprire al traffico di rete, in particolare quando si utilizzano velocità di connessione di rete multigigabit, consentendo a più pacchetti di essere messi in coda affinché il kernel possa elaborarli.

UNIT

Produzione

Garbage Collector Options

- L'ottimizzazione delle opzioni di garbage collection di Java per un'applicazione è sempre stata un'arte, richiedendo informazioni dettagliate su come l'applicazione utilizza la memoria e una quantità significativa di osservazioni, prove ed errori.

- Per fortuna, questo è cambiato con Java 7 e l'introduzione del Garbage First (o G1) Garbage Collector.

- G1 è progettato per adattarsi automaticamente a diversi carichi di lavoro e fornire tempi di pausa coerenti per la garbage collection durante il ciclo di vita dell'applicazione.

- Gestisce facilmente anche grandi dimensioni di heap segmentando l'heap in zone più piccole e non accorpando l'intero heap per ogni pausa.

Garbage Collector Options

- G1 fa tutto questo con una quantità minima di configurazione durante il normale funzionamento.

- Esistono due opzioni di configurazione per G1 utilizzate per regolarne le prestazioni:

  - MaxGCPauseMillis
  - InitiatingHeapOccupancyPercent

Garbage Collector Options: MaxGCPauseMillis

- Questa opzione specifica il tempo di pausa preferito per ciascun ciclo di garbage collection.
- Non è un limite fisso: G1 può e supererà questo tempo se richiesto.
- Questo valore predefinito è 200 millisecondi.
- Ciò significa che G1 tenterà di programmare la frequenza dei cicli GC, nonché il numero di zone raccolte in ciascun ciclo, in modo tale che ciascun ciclo impiegherà circa 200 ms.

Garbage Collector Options: InitiatingHeapOccupancyPercent

- Questa opzione specifica la percentuale dell'heap totale che potrebbe essere in uso prima che G1 inizi un ciclo di raccolta. Il valore predefinito è 45.

- Ciò significa che G1 non avvierà un ciclo di raggruppamento fino a quando non viene utilizzato il 45% dell'heap.

- Ciò include sia il nuovo (Eden) che il vecchio utilizzo della zona in totale.

Garbage Collector Options

- Il broker Kafka è abbastanza efficiente nel modo in cui utilizza la memoria heap quindi è possibile impostare queste opzioni su un valore inferiore.

- Le opzioni di ottimizzazione GC fornite in questa sezione sono risultate appropriate per un server con 64 GB di memoria, che esegue Kafka in un heap da 5 GB.

- Per MaxGCPauseMillis, questo broker può essere configurato con un valore di 20 ms.

- Il valore per InitiatedHeapOccupancyPercent è impostato su 35, il che provoca l'esecuzione della garbage collection leggermente prima rispetto al valore predefinito.

Garbage Collector Options

- La modifica è facile da effettuare tramite le variabili di ambiente.

```
# export JAVA_HOME=/usr/java/jdk1.8.0_51
# export KAFKA_JVM_PERFORMANCE_OPTS="-server -XX:+UseG1GC
-XX:MaxGCPauseMillis=20 -XX:InitiatingHeapOccupancyPercent=35
-XX:+DisableExplicitGC -Djava.awt.headless=true"
# /usr/local/kafka/bin/kafka-server-start.sh -daemon
/usr/local/kafka/config/server.properties
#
```

Datacenter Layout

- Per i sistemi di sviluppo, la posizione fisica dei broker Kafka all'interno di un datacenter non è tanto preoccupante, in quanto non vi è un impatto così grave se il cluster è parzialmente o completamente non disponibile per brevi periodi di tempo.

- Nel servire il traffico di produzione, tuttavia, i tempi di inattività significano la perdita di soldi, sia attraverso la perdita di servizi agli utenti o la perdita di telemetria su ciò che gli utenti stanno facendo.

- Diventa fondamentale configurare la replica all'interno del cluster Kafka così come è importante considerare la posizione fisica dei broker nei loro rack nel datacenter.

- Se non risolto prima della distribuzione di Kafka, potrebbe essere necessaria una costosa manutenzione per spostare i server.

Datacenter Layout

- Il broker Kafka non ha consapevolezza del rack quando assegna nuove partizioni ai broker.

- Ciò significa che non può tenere conto del fatto che due broker possono trovarsi nello stesso rack fisico o nella stessa zona di disponibilità (se in esecuzione in un servizio cloud come AWS), e quindi possono facilmente assegnare tutte le repliche per una partizione ai broker che condividere la stessa potenza e connessioni di rete nello stesso rack.

- Se quel rack dovesse avere un problema, queste partizioni sarebbero offline e inaccessibili ai client. Inoltre, può causare ulteriori perdita di dati sul recupero a causa di un'elezione scorretta del leader.

Datacenter Layout

- La migliore pratica è di avere ciascun broker Kafka in un cluster installato in un rack diverso, o almeno non condividere singoli punti di criticità come alimentazione e rete.

- Ciò significa in genere almeno distribuire i server che eseguiranno broker con connessioni dual power (a due circuiti diversi) e switch a doppia rete (con un'interfaccia collegata sui server stessi per il failover senza soluzione di continuità).

- Anche con connessioni doppie, c'è un vantaggio nell'avere broker in rack completamente separati.

- Di tanto in tanto, potrebbe essere necessario eseguire la manutenzione fisica su un rack o un cabinet che richiede che sia offline (come spostare i server o ricollegare le connessioni di alimentazione).

Collocare le applicazioni su Zookeeper

- Kafka utilizza Zookeeper per la memorizzazione di informazioni sui metadati su broker, argomenti e partizioni.

- Le scritture su Zookeeper vengono eseguite sul cambiamento dei membri dei gruppo di consumer o sulle modifiche al cluster Kafka stesso.

- Questa quantità di traffico è minima e non giustifica l'uso di un ensemble Zookeeper dedicato per un singolo cluster Kafka.

- In effetti, molte distribuzioni useranno un singolo ensemble Zookeeper per più cluster Kafka (usando un percorso chroot Zookeeper per ciascun cluster, come descritto in precedenza).

Collocare le applicazioni su Zookeeper

- Prima di Apache Kafka 0.9.0.0, i consumer, oltre ai broker, utilizzavano Zookeeper per archiviare direttamente le informazioni sulla composizione del gruppo di consumer, gli argomenti che consumava e per committare periodicamente offset per ogni partizione consumata (per abilitare il failover tra i consumatori del gruppo).

- Con la versione 0.9.0.0, è stata introdotta una nuova interfaccia consumer che consente di gestirla direttamente con i broker Kafka.

Collocare le applicazioni su Zookeeper

- Prima di Apache Kafka 0.9.0.0, i consumer, oltre ai broker, utilizzavano Zookeeper per archiviare direttamente le informazioni sulla composizione del gruppo di consumer, gli argomenti che consumava e per committare periodicamente offset per ogni partizione consumata (per abilitare il failover tra i consumatori del gruppo).

- Con la versione 0.9.0.0, è stata introdotta una nuova interfaccia consumer che consente di gestirla direttamente con i broker Kafka.

Collocare le applicazioni su Zookeeper

- Tuttavia, esiste una preoccupazione per i consumer e Zookeeper in determinate configurazioni.

- I consumer hanno una scelta configurabile per utilizzare Zookeeper o Kafka per committare offset e possono anche configurare l'intervallo tra i commit.

- Se il consumatore utilizza Zookeeper per gli offset, ogni consumatore eseguirà una scrittura Zookeeper ad ogni intervallo per ogni partizione che consuma.

- Un intervallo ragionevole per i commit di offset è di 1 minuto, poiché questo è il periodo di tempo durante il quale un gruppo di consumer leggerà i messaggi duplicati in caso di errore del consumer.

Collocare le applicazioni su Zookeeper

- Questi commit possono essere una quantità significativa di traffico Zookeeper, specialmente in un cluster con molti consumer, e dovranno essere presi in considerazione.

- Potrebbe essere necessario utilizzare un intervallo di commit più lungo se l'ensemble Zookeeper non è in grado di gestire il traffico.

- Tuttavia, si consiglia ai consumer che utilizzano le più recenti librerie Kafka di utilizzare Kafka per committare offset rimuovendo la dipendenza da Zookeeper.

- Al di fuori dell'uso di un singolo ensemble per più cluster Kafka, non è consigliabile condividere l'ensemble con altre applicazioni, se è possibile evitarlo.

Collocare le applicazioni su Zookeeper

- Kafka è sensibile alla latenza e ai timeout di Zookeeper e un'interruzione delle comunicazioni con l'ensemble farà sì che i broker si comportino in modo imprevedibile.

- Ciò può facilmente causare la disattivazione simultanea di più broker contemporaneamente (nel caso in cui perdano le connessioni di Zookeeper) causando l'andata offline delle partizioni.

- Inoltre mette sotto stress il controller del cluster, il che può risultare come un errore latente molto tempo dopo l'interruzione, come quando si tenta di eseguire un arresto controllato di un broker.

- Altri problemi possono derivare da applicazioni che possono sollecitare l'ensemble Zookeeper sia per un utilizzo intenso che per operazioni improprie.