Monitoring Kafka

Alex Loddengaard
January 25, 2019 18:08

System

Monitoring the following system metrics:

- CPU usage
- Memory usage
  - Note that the Kafka broker will only use JVM heap space for meta data and buffers. The broker relies on the OS' page cache for caching.
- Available disk space
- Disk IO
- Network IO
- Number of open file handles

Alert when the following is observed:

- 60% disk usage for disks storing the Kafka log (configured via log.dirs)
- 60% disk IO usage
- 60% network IO usage
- 60% file handle usage

Note: 60% is chosen to afford time and resources to add more nodes and move partitions accordingly

Kafka

Kafka publishes JMX metrics, documented here: http://kafka.apache.org/documentation.html#monitoring

The below table has additional information about some broker JMX metrics listed in the documentation:

Metric	Description
kafka.server:type=BrokerTopicMetrics,name=MessagesInPerSec	Number of incoming messages per second. Useful for understanding broker load.
kafka.server:type=BrokerTopicMetrics,name=BytesInPerSec	Bytes in per second. Useful for understanding broker load.
kafka.network:type=RequestMetrics,name=RequestsPerSec,request={Produce/FetchConsumer/FetchFollower}	Number of requests per second, for produce, consumer fetch, and replica follower fetch. Useful for understanding broker load.
kafka.server:type=BrokerTopicMetrics,name=BytesOutPerSec	Bytes out per second. Useful for understanding broker load.
kafka.log:type=LogFlushStats,name=LogFlushRateAndTimeMs	The frequency of Kafka log page cache flushes, and how long the flushes take. Can be used to diagnose a slow disk or misconfigured page cache.
kafka.server:type=ReplicaManager,name=UnderReplicatedPartitions	Should always be 0. If > 0, likely a broker failed. Should alert when not 0.

Metric Description

Recently viewed articles

Confluent Platform Ops Checklist
Monitoring Connect
Monitoring Zookeeper
ZooKeeper Rolling upgrade
Kafka Broker Performance Diagnostics

Related articles

Kafka Broker Performance Diagnostics
Confluent Platform Ops Checklist
Monitoring Zookeeper
Monitoring Connect
Kafka Broker runs out of memory due to maximum mmap count exceeded

kafka.controller:type=KafkaController,name=ActiveControllerCount	Indicates if this broker is the controller. Should alert if this value changes because likely a broker failed.
kafka.controller:type=ControllerStats,name=LeaderElectionRateAndTimeMs	Rate and time of leader election. If non-zero, either a broker has failed, or one or many brokers are experiencing soft failures, causing thrashing.
kafka.server:type=ReplicaManager,name=PartitionCount	Number of replicas on this host. Shouldn't exceed 2-3k.
kafka.server:type=ReplicaManager,name=IsrShrinksPerSec	The rate at which the ISR is shrinking. Like LeaderElectionRateAndTimeMs, the ISR will shrink if a broker is shutdown, either gracefully or not. If the ISR is thrashing, likely brokers are soft failing.
kafka.network:type=RequestMetrics,name=TotalTimeMs,request={Produce/FetchConsumer/FetchFollower}	Total time a request takes to be completed, for produce, consumer fetch, and replica follower fetch. Where request time is spent can be understood by exploring the below metrics.
kafka.network:type=RequestMetrics,name=RequestQueueTimeMs,request={Produce/FetchConsumer/FetchFollower}	The time the request is waiting in the request queue, for produce, consumer fetch, and replica follower fetch. A high value can imply there aren't enough IO threads or the CPU is a bottleneck.
kafka.network:type=RequestMetrics,name=ResponseQueueTimeMs,request={Produce/FetchConsumer/FetchFollower}	The time the request is waiting in the response queue, for produce, consumer fetch, and replica follower fetch. A high value can imply there aren't enough network threads.
kafka.network:type=RequestMetrics,name=LocalTimeMs,request={Produce/FetchConsumer/FetchFollower}	The time the request is being processed by the leader locally, for produce, consumer fetch, and replica follower fetch. Often a high value implies a slow disk.
kafka.network:type=RequestMetrics,name=RemoteTimeMs,request={Produce/FetchConsumer/FetchFollower}	The time the request is waiting on a remote client, for produce, consumer fetch, and replica follower fetch. A high value can imply a slow network connection. For fetch request, if the remote time is high, it could be that there is not enough data to give in a fetch response. This can happen when the consumer or replica is caught up and there is no new incoming data. If this is the case, remote time will be close to the max wait time, which is normal. Max wait time is configured via replica.fetch.wait.max.ms and fetch.max.wait.ms.
kafka.network:type=RequestMetrics,name=ResponseSendTimeMs,request={Produce/FetchConsumer/FetchFollower}	The time the request is being sent back to the client, for produce, consumer fetch, and replica follower fetch. A high value can imply there aren't enough network threads or the CPU or network is a bottleneck.
kafka.network:type=SocketServer,name=NetworkProcessorAvgIdlePercent	The average fraction of time the network threads are idle. Useful when paired with the above request-related JMX metrics.
kafka.server:type=KafkaRequestHandlerPool,name=RequestHandlerAvgIdlePercent	The average fraction of time the I/O threads are idle. Useful when paired with the above request-related JMX metrics.

	related JMX metrics.
kafka.controller:type=KafkaController,name=ControllerState	The state the controller is in, i.e. the event that is currently being processed. Some actions like partition reassignment may take a while and include many events (potentially interleaved with other events), but that doesn't change the fact that at most one event is processed at a time.
kafka.controller:type=ControllerStats,name=ControllerChangeRateAndTimeMs	rate and latency for the controller change state
kafka.controller:type=ControllerStats,name=TopicChangeRateAndTimeMs	rate and latency for the controller to create new topics
kafka.controller:type=ControllerStats,name=TopicDeletionRateAndTimeMs	rate and latency for the controller to delete topics
kafka.controller:type=ControllerStats,name=PartitionReassignmentRateAndTimeMs	rate and latency for the controller to reassign partitions
kafka.controller:type=ControllerStats,name=AutoLeaderBalanceRateAndTimeMs	rate and latency for the controller to auto balance the leaders
kafka.controller:type=ControllerStats,name=ManualLeaderBalanceRateAndTimeMs	rate and latency for the controller to manually balance the leaders
kafka.controller:type=ControllerStats,name=IsrChangeRateAndTimeMs	rate and latency for the controller to manually balance the leaders
kafka.controller:type=ControllerChannelManager,name=TotalQueueSize	total size of the queue in ControllerChannelManager
kafka.controller:type=ControllerChannelManager,name=QueueSize,brokerId=10	QueueSize in ControllerChannelManger for a particular broker
kafka.cluster:type=Partition,name=FailedIsrUpdatesPerSec	measure the occurrences of failed ISR update in ZK
kafka.server:name=UnderMinIsrPartitionCount,type=ReplicaManager	The value of this metric is the total number of leader partitions on this broker whose in-sync replicas < minIsr
kafka.cluster:name=UnderMinIsr,type=Partition,topic={topic},partition={partition}	The value of this metric is 1 if the broker is leader of this partition AND the number of in-sync replicas of this partition < minIsr of this partition. Otherwise it is 0.
kafka.controller:type=KafkaController,name=GlobalTopicCount	count of all topics within the cluster
kafka.controller:type=KafkaController,name=GlobalPartitionCount	count of partitions across all topics in the cluster
*kafka.network:type=RequestMetrics,name=ErrorsPerSec,request=api*key_name,error=error_code_name_	number of errors per second
kafka.server:type=BrokerTopicMetrics,name=FetchMessageConversionsPerSec,topic=([-.\w]+)	number of Fetch messages converted per second for a given topic
kafka.server:type=BrokerTopicMetrics,name=ProduceMessageConversionsPerSec,topic=([-.\w]+)	number of Produce messages converter per second for a given topic

Metric **Description**

kafka.network:type=RequestMetrics,name=MessageConversionsTimeMs,request={Produce or Fetch}	Time in milliseconds spent on message format conversions.
kafka.network:type=RequestMetrics,name=RequestBytes,request=apiKey	Size of requests for each request type.
kafka.network:type=RequestMetrics,name=TemporaryMemoryBytes,request=apiKey	Temporary memory used for message format conversions and decompression.
kafka.server:type=socket-server-metrics,listener=listenerName,networkProcessor=processorIndex	Rate of succerssful/failed authentications using SASL or SSL
kafka.server:type=ZooKeeperClientMetrics,name=ZooKeeperRequestLatencyMs	Latency in millseconds for ZooKeeper requests from broker.
kafka.server:type=SessionExpireListener,name=SessionState	Connection status of broker's ZooKeeper session which may be one of Disconnected
_[kafka.admin.client	kafka.consumer

JVM

JVM garbage collector (GC) logging should be enabled by setting the *GC*LOG_ENABLED_ environment variable to *true*. The GC log will show how long garbage collection takes.

ZooKeeper

ZooKeeper can be monitored in the following ways:

- JMX, documented here: https://zookeeper.apache.org/doc/r3.3.3/zookeeperJMX.html
- Four-letter words, documented here: https://zookeeper.apache.org/doc/r3.3.3/zookeeperAdmin.html#sc_zkCommands
  - In particular, RUOK should return IMOK.
- Monitor the request latency and number of outstanding requests, via the following JMX metrics (created by the ZooKeeper server):
  - On a standalone ZooKeeper server:
    - *org.apache.ZooKeeperService:type=StandaloneServer*port{port-number} AvgRequestLatency_
    - *org.apache.ZooKeeperService:type=StandaloneServer*port{port-number} OutstandingRequests_
  - On a clustered, multi-machine ZooKeeper server:
    - *org.apache.ZooKeeperService:name0=ReplicatedServer*id{myid},name1=replica.{myid},name2=Follower AvgRequestLatency_
    - *org.apache.ZooKeeperService:name0=ReplicatedServer*id{myid},name1=replica.{myid},name2=Follower OutstandingRequests_
- Monitor which ZooKeeper servers are leaders and followers – this is useful when diagnosing issues. This information is published via JMX from the ZooKeeper server.
- Monitor the number of clients and watchers, via the following JMX metrics (created by the ZooKeeper server):
  - On a standalone ZooKeeper server:
    - *org.apache.ZooKeeperService:type=StandaloneServer*port{port-number} NumAliveConnections_
    - *org.apache.ZooKeeperService:type=InMemoryDataTree WatchCount*
  - On a clustered, multi-machine ZooKeeper server:
    - *org.apache.ZooKeeperService:name0=ReplicatedServer*id{myid},name1=replica.{myid},name2=Follower NumAliveConnections_
    - *org.apache.ZooKeeperService:name0=ReplicatedServer*id{myid},name1=replica.{myid},name2=Follower,name3=InMemoryDataTree WatchCount_

Consumer Lag

For "real-time" consumer applications, where the consumer is meant to be processing the newest messages with as little latency as possible, consumer lag should be monitored closely. Most "real-time" applications will want little-to-no consumer lag, because lag introduces end-to-end latency.

Consumer lag can be monitored using the *kafka-consumer-groups* command-line tool, or using the consumer's JMX metric *kafka.consumer:type=consumer-fetch-manager-metrics,client-id={client-id}* attribute: *records-lag-max*.

The *kafka-consumer-groups* command-line tool calculates lag by comparing the last committed offset to the most recent offset, which means lag is a function of when the last commit was made and hence may not be up-to-date. On the other hand, the consumer's records-lag-max JMX metric calculates lag by comparing the offset most recently seen by the consumer to the most recent offset in the log, which is a more real-time measurement.