**Research Activity: Gather matrices for Consumer, Broker & Producer. Write a report about the usage of these.**

A properly functioning Kafka cluster can handle a significant amount of data. It’s important to monitor the health of the Kafka deployment to maintain reliable performance from the applications that depend on it.

Kafka metrics can be broken down into three categories:

* [Producer metrics](https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics/#kafka-producer-metrics)
* [Kafka server (broker) metrics](https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics/#broker-metrics)
* [Consumer metrics](https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics/#kafka-consumer-metrics)
* **Producer matrices**

Producers are responsible for producing data to Kafka topics. If the producer fails, the consumer will not have any new messages to consume and it will be left idle. The performance of the producer also plays an important role in achieving high throughput and latency. Let's look into a few important matrices of Kafka producer:

* **Response rate**: The producer sends records to the Kafka broker, and the broker acknowledges when a message is written to a replica in case of a request. Required .acks is set to -1. The response rate depends on the value assigned to this property. If set to, -0, the broker will immediately return a response when it receives a request from the producer before it writes data to disk. If set to 1, the producer first writes data to its disk and then returns a response. Obviously, less fewer write operations will lead to high performance, but there will be chances of losing data in such cases.
* **Request rate**: The request rate is the number of records the producer produces within a given time.
* **I/O wait time**: The producer sends data and then waits for data. It may wait for network resources when the producing rate is more than the sending rate. The reason for a low producing rate could be slow disk access, and checking the I/O wait time can help us identify the performance of reading the data. More waiting time means producers are not receiving data quickly. In such cases, we may want to use fast access storage such as SSD.
* **Failed send rate**: This gives the number of message requests failed per second. If more messages are failing, it triggers an alarm to find out the root cause of the problem and then fix it.
* **Buffer total bytes**: This represents the maximum memory the producer can use to buffer data before it sends it to brokers. The maximum buffer size will result in high throughput.
* **Compression rate**: This represents the average compression rate for batch records for topic. A higher compression rate triggers us to change the compression type or look for some way to reduce it.
* **Broker matrices**

Brokers are responsible for serving producer and consumer requests. Because all messages must pass through a Kafka broker in order to be consumed, monitoring and alerting on issues as they emerge in the broker cluster is critical. Broker metrics can be broken down into three classes:

* [Kafka-emitted metrics](https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics/#kafka-emitted-metrics)
* [Host-level metrics](https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics/#hostlevel-broker-metrics)
* [JVM garbage collection metrics](https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics/#jvm-garbage-collection-metrics)

### [**Kafka-emitted metrics**](https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics/#kafka-emitted-metrics)

#### [UnderReplicatedPartitions](https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics/#metric-to-watch-underreplicatedpartitions)

In a healthy cluster, the number of in sync replicas (ISRs) should be exactly equal to the total number of replicas. If partition replicas fall too far behind their leaders, the follower partition is removed from the ISR pool, and we should see a corresponding increase in IsrShrinksPerSec. If a broker becomes unavailable, the value of UnderReplicatedPartitions will increase sharply. Since Kafka’s high-availability guarantees cannot be met without replication, investigation is certainly warranted should this metric value exceed zero for extended time periods

#### [IsrShrinksPerSec/IsrExpandsPerSec](https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics/#metric-to-watch-isrshrinkspersecisrexpandspersec)

The number of in-sync replicas (ISRs) for a particular partition should remain fairly static, except when we are expanding the broker cluster or removing partitions. In order to maintain high availability, a healthy Kafka cluster requires a minimum number of ISRs for failover. A replica could be removed from the ISR pool if it has not contacted the leader for some time (configurable with the replica.socket.timeout.ms parameter). We should investigate any flapping in the values of these metrics, and any increase in IsrShrinksPerSec without a corresponding increase in IsrExpandsPerSec shortly thereafter

#### [ActiveControllerCount](https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics/#metric-to-alert-on-activecontrollercount)

The first node to boot in a Kafka cluster automatically becomes the controller, and there can be only one. The controller in a Kafka cluster is responsible for maintaining the list of partition leaders, and coordinating leadership transitions (in the event a partition leader becomes unavailable). If it becomes necessary to replace the controller, ZooKeeper chooses a new controller randomly from the pool of brokers. The sum of ActiveControllerCount across all the brokers should always equal one, and we should alert on any other value that lasts for longer than one second

#### [OfflinePartitionsCount (controller only)](https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics/#metric-to-alert-on-offlinepartitionscount-controller-only)

This metric reports the number of partitions without an active leader. Because all read and write operations are only performed on partition leaders, we should alert on a non-zero value for this metric to prevent service interruptions. Any partition without an active leader will be completely inaccessible, and both consumers and producers of that partition will be blocked until a leader becomes available

#### [LeaderElectionRateAndTimeMs](https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics/#metric-to-watch-leaderelectionrateandtimems)

When a partition leader dies, an election for a new leader is triggered. A partition leader is considered “dead” if it fails to maintain its session with ZooKeeper. Unlike ZooKeeper’s [Zab](https://www.datadoghq.com/pdf/zab.totally-ordered-broadcast-protocol.2008.pdf), Kafka does not employ a majority-consensus algorithm for leadership election. Instead, Kafka’s quorum is composed of the set of all in-sync replicas (ISRs) for a particular partition. Replicas are considered in-sync if they are caught-up to the leader, which means that any replica in the ISR can be promoted to the leader.

LeaderElectionRateAndTimeMs reports the rate of leader elections (per second) and the total time the cluster went without a leader (in milliseconds). Although not as bad as UncleanLeaderElectionsPerSec, we will want to keep an eye on this metric. As mentioned above, a leader election is triggered when contact with the current leader is lost, which could translate to an offline broker.

### [**Host-level broker metrics**](https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics/#host-level-broker-metrics)

#### [Page cache read ratio](https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics/#metric-to-watch-page-cache-read-ratio)

Kafka was designed [from the beginning](https://kafka.apache.org/documentation.html#persistence) to leverage the kernel’s page cache in order to provide a reliable (disk-backed) and performant (in-memory) message pipeline. The page cache read ratio is similar to [cache-hit ratio](https://www.datadoghq.com/blog/how-to-monitor-redis-performance-metrics/#metric-to-watch-hit-rate) in databases—a higher value equates to faster reads and thus better performance. This metric will drop briefly if a replica is catching up to a leader (as when a new broker is spawned), but if the page cache read ratio remains below 80 percent, we may benefit from provisioning additional brokers.

#### [Disk usage](https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics/#metric-to-alert-on-disk-usage)

Because Kafka [persists all data to disk](https://kafka.apache.org/documentation.html#persistence), it is necessary to monitor the amount of free disk space available to Kafka. Kafka will fail should its disk become full, so it’s very important that we keep track of disk growth over time, and set alerts to inform administrators at an appropriate amount of time before disk space is all but used up.

#### [CPU usage](https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics/#metric-to-watch-cpu-usage)

Although Kafka’s primary bottleneck is usually memory, it doesn’t hurt to keep an eye on its CPU usage. Even in use cases where GZIP compression is enabled, the CPU is [rarely the source](https://cwiki.apache.org/confluence/display/KAFKA/Operations#Operations-Hardware) of a performance problem. Therefore, if we do see spikes in CPU utilization, it is worth investigating

### [**JVM garbage collection metrics**](https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics/#jvm-garbage-collection-metrics)

Because Kafka is written in Scala and runs in the Java Virtual Machine (JVM), it relies on Java garbage collection processes to free up memory. The more activity in the Kafka cluster, the more often the garbage collection will run.

#### [Young generation garbage collection time](https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics/#metric-to-watch-young-generation-garbage-collection-time)

Young-generation garbage collection occurs relatively often. This is stop-the-world garbage collection, meaning that all application threads pause while it’s carried out. Any significant increase in the value of this metric will dramatically impact Kafka’s performance.

#### [Old generation garbage collection count/time](https://www.datadoghq.com/blog/monitoring-kafka-performance-metrics/#metric-to-watch-old-generation-garbage-collection-counttime)

Old generation garbage collection frees up unused memory in the old generation of the heap. This is low-pause garbage collection, meaning that although it does temporarily stop application threads, it does so only intermittently. If this process is taking a few seconds to complete, or is occurring with increased frequency, the cluster may not have enough memory to function efficiently

* **Consumer metrics**

Consumers are responsible for consuming data from topic and doing some processing on it, if needed. Sometimes, the consumer may be slow, or it may behave unacceptably. The following are some important metrics that will help us identify some parameters that indicate optimization on the consumer side:

* **records-lag-max**: The calculated difference between the producer's current offset and the consumer's current offset is known as record lag. If the difference is very big, it's fairly indicative of the consumer processing data much slower than the producer. It sends alerts for suitable action to fix up this issue, either by adding more consumer instance or by increasing partitions and increasing consumers simultaneously.
* **bytes-consumed-rate:**This represents the number of bytes consumed per second by the consumer. It helps in identifying the network bandwidth of the consumer.
* **records-consumed-rate**: This defines the number of messages consumed per second. This value should be constant and generally helps when compared with byte-consumed-rate.
* **fetch-rate**: This represents the number of records fetched per second by the consumer.
* **fetch-latency-max**: This represents the maximum time taken for the fetch request. If it's high, it triggers to optimize the consumer application.