Sizing Amazon OpenSearch Service domains

PDF (/pdfs/opensearch-service/latest/developerguide/opensearch-service-dg.pdf#sizing-domains) | **RSS (amazon-opensearch-service-release-notes.rss)**

There's no perfect method of sizing Amazon OpenSearch Service domains. However, by starting with an understanding of your storage needs, the service, and OpenSearch itself, you can make an educated initial estimate on your hardware needs. This estimate can serve as a useful starting point for the most critical aspect of sizing domains: testing them with representative workloads and monitoring their performance.

Topics

Calculating storage requirements

Most OpenSearch workloads fall into one of two broad categories:

- **Long-lived index**: You write code that processes data into one or more OpenSearch indexes and then updates those indexes periodically as the source data changes. Some common examples are website, document, and ecommerce search.
- **Rolling indexes**: Data continuously flows into a set of temporary indexes, with an indexing period and retention window (such as a set of daily indexes that is retained for two weeks). Some common

examples are log analytics, time-series processing, and clickstream analytics.

For long-lived index workloads, you can examine the source data on disk and easily determine how much storage space it consumes. If the data comes from multiple sources, just add those sources together.

For rolling indexes, you can multiply the amount of data generated during a representative time period by the retention period. For example, if you generate 200 MiB of log data per hour, that's 4.7 GiB per day, which is 66 GiB of data at any given time if you have a two-week retention period.

The size of your source data, however, is just one aspect of your storage requirements. You also have to consider the following:

- **Number of replicas**: Each replica is a full copy of an index and needs the same amount of disk space. By default, each OpenSearch index has one replica. We recommend at least one to prevent data loss. Replicas also improve search performance, so you might want more if you have a read-heavy workload. Use `PUT /my-index/_settings` to update the `number_of_replicas` setting for your index.
- **OpenSearch indexing overhead**: The on-disk size of an index varies. The total size of the source data plus the index is often 110% of the source, with the index up to 10% of the source data. After you index your data, you can use the `_cat/indices?v` API and `pri.store.size` value to calculate the exact overhead. `_cat/allocation?v` also provides a useful summary.
- **Operating system reserved space**: By default, Linux reserves 5% of the file system for the `root` user for critical processes, system recovery, and to safeguard against disk fragmentation problems.
- **OpenSearch Service overhead**: OpenSearch Service reserves 20% of the storage space of each instance (up to 20 GiB) for segment merges, logs, and other internal operations.

  Because of this 20 GiB maximum, the total amount of reserved space can vary dramatically depending on the number of instances in your domain. For example, a domain might have three `m6g.xlarge.search` instances, each with 500 GiB of storage space, for a total of 1.46 TiB. In this case, the total reserved space is only 60 GiB. Another domain might have 10 `m3.medium.search` instances, each with 100 GiB of storage space, for a total of 0.98 TiB. Here, the

total reserved space is 200 GiB, even though the first domain is 50% larger.

In the following formula, we apply a "worst-case" estimate for overhead. This estimate includes additional free space to help minimize the impact of node failures and Availability Zone outages.

In summary, if you have 66 GiB of data at any given time and want one replica, your *minimum* storage requirement is closer to 66 * 2 * 1.1 / 0.95 / 0.8 = 191 GiB. You can generalize this calculation as follows:

Source data * (1 + number of replicas) * (1 + indexing overhead) / (1 - Linux reserved space) / (1 - OpenSearch Service overhead) = minimum storage requirement

Or you can use this simplified version:

Source data * (1 + number of replicas) * 1.45 = minimum storage requirement

Insufficient storage space is one of the most common causes of cluster instability. So you should cross-check the numbers when you choose instance types, instance counts, and storage volumes (#bp-instances) .

Other storage considerations exist:

- If your minimum storage requirement exceeds 1 PB, see Petabyte scale in Amazon OpenSearch Service (./petabyte-scale.html) .
- If you have rolling indexes and want to use a hot-warm architecture, see UltraWarm storage for Amazon OpenSearch Service (./ultrawarm.html) .

Choosing the number of shards

After you understand your storage requirements, you can investigate your indexing strategy. By default in OpenSearch Service, each index is divided into five primary shards and one replica (total of 10 shards). This behavior differs from open source OpenSearch, which defaults to one primary and one replica shard. Because you can't easily change the number of primary shards for an existing index, you should decide about shard count *before* indexing your first document.

The overall goal of choosing a number of shards is to distribute an index evenly across all data nodes in the cluster. However, these shards shouldn't be too large or too numerous. A general guideline is to try to keep shard size between 10–30 GiB for workloads where search latency is a key performance objective, and 30–50 GiB for write-heavy workloads such as log analytics.

Large shards can make it difficult for OpenSearch to recover from failure, but because each shard uses some amount of CPU and memory, having too many small shards can cause performance issues and out of memory errors. In other words, shards should be small enough that the underlying OpenSearch Service instance can handle them, but not so small that they place needless strain on the hardware.

For example, suppose you have 66 GiB of data. You don't expect that number to increase over time, and you want to keep your shards around 30 GiB each. Your number of shards therefore should be approximately 66 * 1.1 / 30 = 3. You can generalize this calculation as follows:

(Source data + room to grow) * (1 + indexing overhead) / desired shard size = approximate number of primary shards

This equation helps compensate for data growth over time. If you expect those same 66 GiB of data to quadruple over the next year, the approximate number of shards is (66 + 198) * 1.1 / 30 = 10. Remember, though, you don't have those extra 198 GiB of data *yet*. Check to make sure that this preparation for the future doesn't create unnecessarily tiny shards that consume huge amounts of CPU and memory in the present. In this case, 66 * 1.1 / 10 shards = 7.26 GiB per shard, which will consume extra resources and is below the recommended size range. You might consider the more middle-of-the-road approach of six shards, which leaves you with 12-GiB shards today and 48-GiB shards in the future. Then again, you might prefer to start with three shards and reindex your data when the shards exceed 50 GiB.

A far less common issue involves limiting the number of shards per node. If you size your shards appropriately, you typically run out of disk space long before encountering this limit. For example, an `m6g.large.search` instance has a maximum disk size of 512 GiB. If you stay below 80% disk usage and size your shards at 20 GiB, it can accommodate approximately 20 shards. Elasticsearch 7.*x* and later, and all versions of OpenSearch, have

a limit of *1,000* shards per node. To adjust the maximum shards per node, configure the `cluster.max_shards_per_node` setting. For an example, see Cluster settings (https://opensearch.org/docs/latest/opensearch/rest-api/cluster-settings/#request-body) .

Sizing shards appropriately almost always keeps you below this limit, but you can also consider the number of shards for each GiB of Java heap. On a given node, have no more than 25 shards per GiB of Java heap. For example, an `m5.large.search` instance has a 4-GiB heap, so each node should have no more than 100 shards. At that shard count, each shard is roughly 5 GiB in size, which is well below our recommendation.

Choosing instance types and testing

After you calculate your storage requirements and choose the number of shards that you need, you can start to make hardware decisions. Hardware requirements vary dramatically by workload, but we can still offer some basic recommendations.

In general, the storage limits (./limits.html) for each instance type map to the amount of CPU and memory that you might need for light workloads. For example, an `m6g.large.search` instance has a maximum EBS volume size of 512 GiB, 2 vCPU cores, and 8 GiB of memory. If your cluster has many shards, performs taxing aggregations, updates documents frequently, or processes a large number of queries, those resources might be insufficient for your needs. If your cluster falls into one of these categories, try starting with a configuration closer to 2 vCPU cores and 8 GiB of memory for every 100 GiB of your storage requirement.

> **Tip**
>
> For a summary of the hardware resources that are allocated to each instance type, see Amazon OpenSearch Service pricing (https://aws.amazon.com/opensearch-service/pricing/) .

Still, even those resources might be insufficient. Some OpenSearch users report that they need many times those resources to fulfill their requirements. To find the right hardware for your workload, you have to make an educated initial estimate, test with representative workloads,

adjust, and test again.

Step 1: Make an initial estimate

To start, we recommend a minimum of three nodes to avoid potential OpenSearch issues, such as a *split brain* state (when a lapse in communication leads to a cluster having two master nodes). If you have three dedicated master nodes (./managedomains-dedicatedmasternodes.html) , we still recommend a minimum of two data nodes for replication.

Step 2: Calculate storage requirements per node

If you have a 184-GiB storage requirement and the recommended minimum number of three nodes, use the equation 184 / 3 = 61 GiB to find the amount of storage that each node needs. In this example, you might select three `m6g.large.search` instances, where each uses a 90-GiB EBS storage volume, so that you have a safety net and some room for growth over time. This configuration provides 6 vCPU cores and 24 GiB of memory, so it's suited to lighter workloads.

For a more substantial example, consider a 14 TiB (14,336 GiB) storage requirement and a heavy workload. In this case, you might choose to begin testing with 2 * 144 = 288 vCPU cores and 8 * 144 = 1152 GiB of memory. These numbers work out to approximately 18 `i3.4xlarge.search` instances. If you don't need the fast, local storage, you could also test 18 `r6g.4xlarge.search` instances, each using a 1-TiB EBS storage volume.

If your cluster includes hundreds of terabytes of data, see Petabyte scale in Amazon OpenSearch Service (./petabyte-scale.html) .

Step 3: Perform representative testing

After configuring the cluster, you can add your indexes (./indexing.html) using the number of shards you calculated earlier, perform some representative client testing using a realistic dataset, and monitor CloudWatch metrics (./managedomains-cloudwatchmetrics.html) to see how the cluster handles the workload.

Step 4: Succeed or iterate

If performance satisfies your needs, tests succeed, and CloudWatch metrics are normal, the cluster is ready to use. Remember to set CloudWatch alarms (./cloudwatch-alarms.html) to detect unhealthy resource usage.

If performance isn't acceptable, tests fail, or `CPUUtilization` or `JVMMemoryPressure` are high, you might need to choose a different instance type (or add instances) and continue testing. As you add instances, OpenSearch automatically rebalances the distribution of shards throughout the cluster.

Because it's easier to measure the excess capacity in an overpowered cluster than the deficit in an underpowered one, we recommend starting with a larger cluster than you think you need. Next, test and scale down to an efficient cluster that has the extra resources to ensure stable operations during periods of increased activity.

Production clusters or clusters with complex states benefit from dedicated master nodes (./managedomains-dedicatedmasternodes.html) , which improve performance and cluster reliability.