

Operational best practices for Amazon OpenSearch Service

[PDF \(/pdfs/opensearch-service/latest/developerguide/opensearch-service-dg.pdf#bp\)](#)

[RSS \(amazon-opensearch-service-release-notes.rss\)](#)

This chapter provides best practices for operating Amazon OpenSearch Service domains and includes general guidelines that apply to many use cases. Each workload is unique, with unique characteristics, so no generic recommendation is exactly right for every use case. The most important best practice is to deploy, test, and tune your domains in a continuous cycle to find the optimal configuration, stability, and cost for your workload.

Topics

- [Monitoring and alerting \(#bp-monitoring\)](#)
- [Shard strategy \(#bp-sharding-strategy\)](#)
- [Stability \(#bp-stability\)](#)
- [Performance \(#bp-perf\)](#)
- [Security \(#bp-security\)](#)
- [Cost optimization \(#bp-cost-optimization\)](#)
- [Sizing Amazon OpenSearch Service domains \(./sizing-domains.html\)](#)
- [Petabyte scale in Amazon OpenSearch Service \(./petabyte-scale.html\)](#)
- [Dedicated master nodes in Amazon OpenSearch Service \(./manageddomains-dedicatedmasternodes.html\)](#)
- [Recommended CloudWatch alarms for Amazon OpenSearch Service \(./cloudwatch-alarms.html\)](#)


Monitoring and alerting

The following best practices apply to monitoring your OpenSearch Service domains.

Configure CloudWatch alarms

OpenSearch Service emits performance metrics to Amazon CloudWatch. Regularly review your [cluster and instance metrics \(./manageddomains-cloudwatchmetrics.html\)](#) and configure [recommended CloudWatch alarms \(./cloudwatch-alarms.html\)](#) based on your workload performance.

Enable log publishing

OpenSearch Service exposes OpenSearch error logs, search slow logs, indexing slow logs, and audit logs in Amazon CloudWatch Logs. Search slow logs, indexing slow logs, and error logs are useful for troubleshooting performance and stability issues. Audit logs, which are only available if you enable [fine-grained access control \(./fgac.html\)](#) to track user activity. For more information, see [Logs](#)  (<https://opensearch.org/docs/latest/monitoring-your-cluster/logs/>) in the OpenSearch documentation.

Search slow logs and indexing slow logs are an important tool for understanding and troubleshooting the performance of your search and indexing operations. [Enable search and index slow log delivery \(./createdomain-configure-slow-logs.html#createdomain-configure-slow-logs-console\)](#) for all production domains. You must also [configure logging thresholds \(./createdomain-configure-slow-logs.html#createdomain-configure-slow-logs-indices\)](#) —otherwise, CloudWatch won't capture the logs.

Shard strategy

Shards distribute your workload across the data nodes in your OpenSearch Service domain. Properly configured indexes can help boost overall domain performance.

When you send data to OpenSearch Service, you send that data to an index. An index is analogous to a database table, with *documents* as the rows, and *fields* as the columns. When you create the index, you tell OpenSearch how many primary shards you want to create. The primary shards are independent partitions of the full dataset. OpenSearch Service automatically distributes your data across the primary shards in an index. You can also configure *replicas* of the index. Each replica shard comprises a full set of copies of the primary shards for that index.

OpenSearch Service maps the shards for each index across the data nodes in your cluster. It ensures that the primary and replica shards for the index reside on different data nodes. The first replica ensures that you have two copies of the data in the index. You should always use at least one replica. Additional replicas provide additional redundancy and read capacity.

OpenSearch sends indexing requests to all of the data nodes that contain shards that belong to the index. It sends indexing requests first to data nodes that contain primary shards, and

then to data nodes that contain replica shards. Search requests are routed by the coordinator node to either a primary or replica shard for all shards belonging to the index.

For example, for an index with five primary shards and one replica, each indexing request touches 10 shards. In contrast, search requests are sent to n shards, where n is the number of primary shards. For an index with five primary shards and one replica, each search query touches five shards (primary or replica) from that index.

Determine shard and data node counts

Use the following best practices to determine shard and data node counts for your domain.

Shard size – The size of data on disk is a direct result of the size of your source data, and it changes as you index more data. The source-to-index ratio can vary wildly, from 1:10 to 10:1 or more, but usually it's around 1:1.10. You can use that ratio to predict the index size on disk. You can also index some data and retrieve the actual index sizes to determine the ratio for your workload. After you have a predicted index size, set a shard count so that each shard will be between 10–30 GiB (for search workloads), or between 30–50 GiB (for logs workloads). 50 GiB should be the maximum—be sure to plan for growth.

Shard count – The distribution of shards to data nodes has a large impact on a domain's performance. When you have indexes with multiple shards, try to make the shard count an even multiple of the data node count. This helps to ensure that shards are evenly distributed across data nodes, and prevents hot nodes. For example, if you have 12 primary shards, your data node count should be 2, 3, 4, 6, or 12. However, shard count is secondary to shard size—if you have 5 GiB of data, you should still use a single shard.

Shards per data node – The total number of shards that a node can hold is proportional to the node's Java virtual machine (JVM) heap memory. Aim for 25 shards or fewer per GiB of heap memory. For example, a node with 32 GiB of heap memory should hold no more than 800 shards. Although shard distribution can vary based on your workload patterns, there's a limit of 1,000 shards per node. The [cat/allocation](https://opensearch.org/docs/latest/api-reference/cat/cat-allocation/) API provides a quick view of the number of shards and total shard storage across data nodes.

Shard to CPU ratio – When a shard is involved in an indexing or search request, it uses a vCPU to process the request. As a best practice, use an initial scale point of 1.5 vCPU per shard. If your instance type has 8 vCPUs, set your data node count so that each node has no more than six shards. Note that this is an approximation. Be sure to test your workload and scale your cluster accordingly.

For storage volume, shard size, and instance type recommendations, see the following resources:

- [Sizing Amazon OpenSearch Service domains \(./sizing-domains.html\)](#)

- [Petabyte scale in Amazon OpenSearch Service \(./petabyte-scale.html\)](#)

Avoid storage skew

Storage skew occurs when one or more nodes within a cluster holds a higher proportion of storage for one or more indexes than the others. Indications of storage skew include uneven CPU utilization, intermittent and uneven latency, and uneven queueing across data nodes. To determine whether you have skew issues, see the following troubleshooting sections:

- [Node shard and storage skew \(./handling-errors.html#handling-errors-node-skew\)](#)
- [Index shard and storage skew \(./handling-errors.html#handling-errors-index-skew\)](#)

Stability

The following best practices apply to maintaining a stable and healthy OpenSearch Service domain.

Keep current with OpenSearch

Service software updates

OpenSearch Service regularly releases [software updates \(./service-software.html\)](#) that add features or otherwise improve your domains. Updates don't change the OpenSearch or Elasticsearch engine version. We recommend that you schedule a recurring time to run the [DescribeDomain \(https://docs.aws.amazon.com/opensearch-service/latest/APIReference/API_DescribeDomain.html\)](#) API operation, and initiate a service software update if the `UpdateStatus` is `ELIGIBLE`. If you don't update your domain within a certain time frame (typically two weeks), OpenSearch Service automatically performs the update.

OpenSearch version upgrades

OpenSearch Service regularly adds support for community-maintained versions of OpenSearch. Always upgrade to the latest OpenSearch versions when they're available.

OpenSearch Service simultaneously upgrades both OpenSearch and OpenSearch Dashboards (or Elasticsearch and Kibana if your domain is running a legacy engine). If the cluster has dedicated master nodes, upgrades complete without downtime. Otherwise, the cluster might be unresponsive for several seconds post-upgrade while it elects a master node. OpenSearch Dashboards might be unavailable during some or all of the upgrade.



There are two ways to upgrade a domain:

- [In-place upgrade \(./version-migration.html#starting-upgrades\)](#) – This option is easier because you keep the same cluster.

- [Snapshot/restore upgrade \(./version-migration.html#snapshot-based-migration\)](#) – This option is good for testing new versions on a new cluster or migrating between clusters.

Regardless of which upgrade process you use, we recommend that you maintain a domain that is solely for development and testing, and upgrade it to the new version *before* you upgrade your production domain. Choose **Development and testing** for the deployment type when you're creating the test domain. Make sure to upgrade all clients to compatible versions immediately following the domain upgrade.

Back up your data

You can take manual snapshots for cluster recovery, or to move data from one cluster to another. You have to initiate or schedule manual snapshots. Snapshots are stored in your own Amazon S3 bucket. For instructions on how to take and restore a snapshot, see [Creating index snapshots in Amazon OpenSearch Service \(./manageddomains-snapshots.html\)](#) .

Enable dedicated master nodes

[Dedicated master nodes \(./manageddomains-dedicatedmasternodes.html\)](#) improve cluster stability. A dedicated master node performs cluster management tasks, but doesn't hold index data or respond to client requests. This offloading of cluster management tasks increases the stability of your domain and makes it possible for some [configuration changes \(./manageddomains-configuration-changes.html\)](#) to happen without downtime.

Enable and use three dedicated master nodes for optimal domain stability across three Availability Zones. Deploying with [Multi-AZ with Standby \(https://docs.aws.amazon.com/opensearch-service/latest/developerguide/manageddomains-multiaz.html#manageddomains-za-standby\)](#) configures three dedicated master nodes for you. For instance type recommendations, see [Choosing instance types for dedicated master nodes \(./manageddomains-dedicatedmasternodes.html#dedicatedmasternodes-instance\)](#) .

Deploy across multiple Availability Zones

To prevent data loss and minimize cluster downtime in the event of a service disruption, you can distribute nodes across two or three [Availability Zones \(./manageddomains-multiaz.html\)](#) in the same AWS Region. Best practice is to deploy using [Multi-AZ with Standby \(https://docs.aws.amazon.com/opensearch-service/latest/developerguide/manageddomains-multiaz.html#manageddomains-za-standby\)](#) , which configures three Availability Zones, with two zones active and one acting as a standby, and with two replica shards per index. This configuration lets OpenSearch Service distribute replica shards to different AZs than their corresponding primary shards. There are no cross-AZ data transfer charges for cluster communications between Availability Zones.

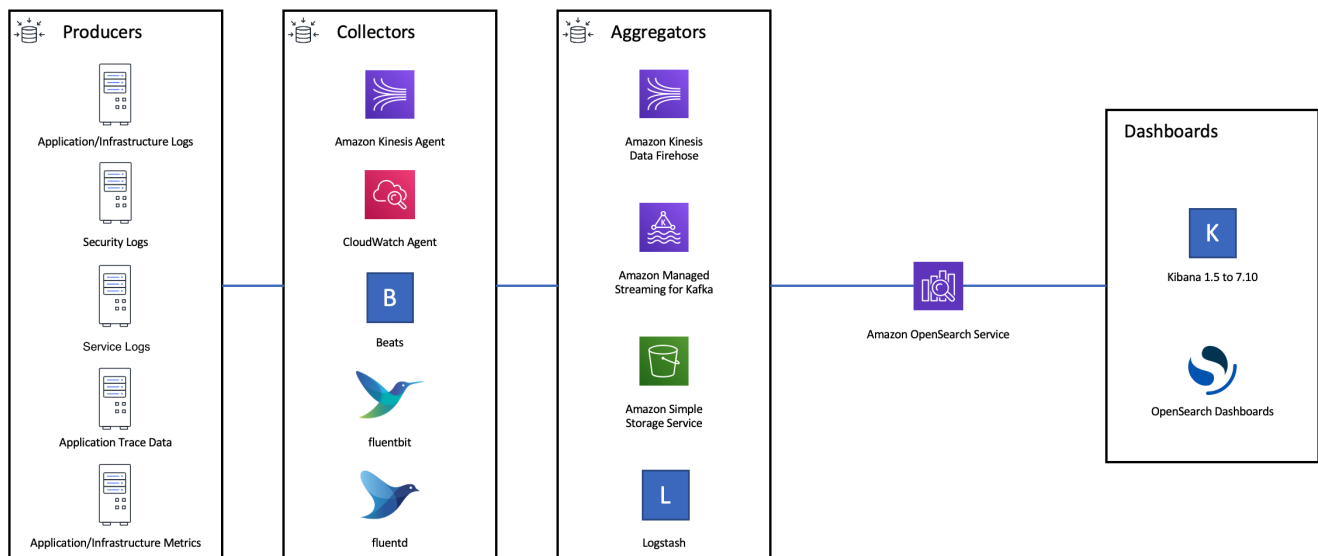
Availability Zones are isolated locations within each Region. With a two-AZ configuration, losing one Availability Zone means that you lose half of all domain capacity. Moving to three Availability Zones further reduces the impact of losing a single Availability Zone.

Control ingest flow and buffering

We recommend that you limit the overall request count using the [_bulk](https://opensearch.org/docs/latest/api-reference/document-apis/bulk/) API operation. It's more efficient to send one `_bulk` request that contains 5,000 documents than it is to send 5,000 requests that contain a single document.

For optimal operational stability, it's sometimes necessary to limit or even pause the upstream flow of indexing requests. Limiting the rate of index requests is an important mechanism for dealing with unexpected or occasional spikes in requests that might otherwise overwhelm the cluster. Consider building a flow control mechanism into your upstream architecture.

The following diagram shows multiple component options for a log ingest architecture. Configure the aggregation layer to allow sufficient space to buffer incoming data for sudden traffic spikes and brief domain maintenance.



Create mappings for search workloads

For search workloads, create [mappings](https://opensearch.org/docs/latest/field-types/index/) that define how OpenSearch stores and indexes documents and their fields. Set `dynamic` to `strict` in order to prevent new fields from being added accidentally.

```
PUT my-index
{
```

```
"mappings": {
  "dynamic": "strict",
  "properties": {
    "title": { "type" : "text" },
    "author": { "type" : "integer" },
    "year": { "type" : "text" }
  }
}
```

Use index templates

You can use an [index template](https://opensearch.org/docs/latest/opensearch/index-templates/) (https://opensearch.org/docs/latest/opensearch/index-templates/) as a way to tell OpenSearch how to configure an index when it's created. Configure index templates before creating indexes. Then, when you create an index, it inherits the settings and mappings from the template. You can apply more than one template to a single index, so you can specify settings in one template and mappings in another. This strategy allows one template for common settings across multiple indexes, and separate templates for more specific settings and mappings.

The following settings are helpful to configure in templates:

- Number of primary and replica shards
- Refresh interval (how often to refresh and make recent changes to the index available to search)
- Dynamic mapping control
- Explicit field mappings

The following example template contains each of these settings:

```
{
  "index_patterns": [
    "index-*"
  ],
  "order": 0,
  "settings": {
    "index": {
      "number_of_shards": 3,
      "number_of_replicas": 1,
      "refresh_interval": "60s"
    }
  },
  "mappings": {
```

```
"dynamic": false,
"properties": {
  "field_name1": {
    "type": "keyword"
  }
}
}
```

Even if they rarely change, having settings and mappings defined centrally in OpenSearch is simpler to manage than updating multiple upstream clients.

Manage indexes with Index State Management

If you're managing logs or time-series data, we recommend using [Index State Management \(ISM\)](#) ([./ism.html](#)). ISM lets you automate regular index lifecycle management tasks. With ISM, you can create policies that invoke index alias rollovers, take index snapshots, move indexes between storage tiers, and delete old indexes. You can even use the ISM [rollover](#) (<https://opensearch.org/docs/latest/im-plugin/ism/policies/#rollover>) operation as an alternative data lifecycle management strategy to avoid shard skew.

First, set up an ISM policy. For example, see [Sample policies \(./ism.html#ism-example\)](#). Then, attach the policy to one or more indexes. If you include an [ISM template \(./ism.html#ism-template\)](#) field in the policy, OpenSearch Service automatically applies the policy to any index that matches the specified pattern.

Remove unused indexes

Regularly review the indexes in your cluster and identify any that aren't in use. Take a snapshot of those indexes so that they're stored in S3, and then delete them. When you remove unused indexes, you reduce the shard count, and make it possible to have more balanced storage distribution and resource utilization across nodes. Even when they're idle, indexes consume some resources during internal index maintenance activities.

Rather than manually deleting unused indexes, you can use ISM to automatically take a snapshot and delete indexes after a certain period of time.

Use multiple domains for high availability

To achieve high availability beyond [99.9% uptime](#) (<https://aws.amazon.com/opensearch-service/sla/>) across multiple Regions, consider using two domains. For small or slowly changing datasets, you can set up [cross-cluster replication \(./replication.html\)](#) to maintain an active-passive model. In this model, only the leader domain is written to, but either domain can be

read from. For larger data sets and quickly changing data, configure dual delivery in your ingest pipeline so that all data is written independently to both domains in an active-active model.

Architect your upstream and downstream applications with failover in mind. Make sure to test the failover process along with other disaster recovery processes.

Performance

The following best practices apply to tuning your domains for optimal performance.

Optimize bulk request size and compression

Bulk sizing depends on your data, analysis, and cluster configuration, but a good starting point is 3–5 MiB per bulk request.

Send requests and receive responses from your OpenSearch domains by using [gzip compression \(./gzip.html\)](#) to reduce the payload size of requests and responses. You can use gzip compression with the [OpenSearch Python client \(./gzip.html#gzip-code\)](#), or by including the following [headers \(./gzip.html#gzip-headers\)](#) from the client side:

- 'Accept-Encoding': 'gzip'
- 'Content-Encoding': 'gzip'

To optimize your bulk request sizes, start with a bulk request size of 3 MiB. Then, slowly increase the request size until indexing performance stops improving.

Note

To enable gzip compression on domains running Elasticsearch version 6.x, you must set `http_compression.enabled` at the cluster level. This setting is true by default in Elasticsearch versions 7.x and all versions of OpenSearch.

Reduce the size of bulk request responses

To reduce the size of OpenSearch responses, exclude unnecessary fields with the `filter_path` parameter. Make sure that you don't filter out any fields that are required to identify or retry failed requests. For more information and examples, see [Reducing response size \(./indexing.html#indexing-size\)](#).

Tune refresh intervals

OpenSearch indexes have eventual read consistency. A refresh operation makes all the updates that are performed on an index available for search. The default refresh interval is one second, which means that OpenSearch performs a refresh every second while an index is being written to.

The less frequently that you refresh an index (higher refresh interval), the better the overall indexing performance is. The trade-off of increasing the refresh interval is that there's a longer delay between an index update and when the new data is available for search. Set your refresh interval as high as you can tolerate to improve overall performance.

We recommend setting the `refresh_interval` parameter for all of your indexes to 30 seconds or more.

Enable Auto-Tune

[Auto-Tune \(./auto-tune.html\)](#) uses performance and usage metrics from your OpenSearch cluster to suggest changes to queue sizes, cache sizes, and Java virtual machine (JVM) settings on your nodes. These optional changes improve cluster speed and stability. You can revert to the default OpenSearch Service settings at any time. Auto-Tune is enabled by default on new domains unless you explicitly disable it.

We recommend that you enable Auto-Tune on all domains, and either set a recurring maintenance window or periodically review its recommendations.

Security

The following best practices apply to securing your domains.

Enable fine-grained access control

[Fine-grained access control \(./fgac.html\)](#) lets you control who can access certain data within an OpenSearch Service domain. Compared to generalized access control, fine-grained access control gives each cluster, index, document, and field its own specified policy for access. Access criteria can be based on a number of factors, including the role of the person who is requesting access and the action that they intend to perform on the data. For example, you might give one user access to write to an index, and another user access only to read the data on the index without making any changes.

Fine-grained access control allows data with different access requirements to exist in the same storage space without running into security or compliance issues.

We recommend enabling fine-grained access control on your domains.

Deploy domains within a VPC

Placing your OpenSearch Service domain within a virtual private cloud (VPC) helps enable secure communication between OpenSearch Service and other services within the VPC—without the need for an internet gateway, NAT device, or VPN connection. All traffic remains securely within the AWS Cloud. Because of their logical isolation, domains that reside within a VPC have an extra layer of security compared to domains that use public endpoints.

We recommend that you [create your domains within a VPC \(./vpc.html\)](#).

Apply a restrictive access policy

Even if your domain is deployed within a VPC, it's a best practice to implement security in layers. Make sure to [check the configuration \(./createupdatedomains.html#createdomain-configure-access-policies\)](#) of your current access policies.

Apply a restrictive [resource-based access policy \(./ac.html#ac-types-resource\)](#) to your domains and follow the [principle of least privilege \(https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#grant-least-privilege\)](#) when granting access to the configuration API and the OpenSearch API operations. As a general rule, avoid using the anonymous user principal `"Principal": {"AWS": "*" }` in your access policies.

There are some situations, however, where it's acceptable to use an open access policy, such as when you enable fine-grained access control. An open access policy can enable you to access the domain in cases where request signing is difficult or impossible, such as from certain clients and tools.

Enable encryption at rest

OpenSearch Service domains offer encryption of data at rest to help prevent unauthorized access to your data. Encryption at rest uses AWS Key Management Service (AWS KMS) to store and manage your encryption keys, and the Advanced Encryption Standard algorithm with 256-bit keys (AES-256) to perform the encryption.

If your domain stores sensitive data, [enable encryption of data at rest \(./encryption-at-rest.html\)](#).

Enable node-to-node encryption

Node-to-node encryption provides an additional layer of security on top of the default security features within OpenSearch Service. It implements Transport Layer Security (TLS) for all communications between the nodes that are provisioned within OpenSearch. Node-to-node encryption, any data sent to your OpenSearch Service domain over HTTPS remains encrypted in transit while it's being distributed and replicated between nodes.

If your domain stores sensitive data, [enable node-to-node encryption \(./ntn.html\)](#).

Monitor with AWS Security Hub

Monitor your usage of OpenSearch Service as it relates to security best practices by using [AWS Security Hub](https://docs.aws.amazon.com/securityhub/latest/userguide/what-is-securityhub.html) (<https://docs.aws.amazon.com/securityhub/latest/userguide/what-is-securityhub.html>) . Security Hub uses security controls to evaluate resource configurations and security standards to help you comply with various compliance frameworks. For more information about using Security Hub to evaluate OpenSearch Service resources, see [Amazon OpenSearch Service controls](https://docs.aws.amazon.com/securityhub/latest/userguide/opensearch-controls.html) (<https://docs.aws.amazon.com/securityhub/latest/userguide/opensearch-controls.html>) in the *AWS Security Hub User Guide*.

Cost optimization

The following best practices apply to optimizing and saving on your OpenSearch Service costs.

Use the latest generation instance types

OpenSearch Service is always adopting new Amazon EC2 [instances types](#) ([./supported-instance-types.html](#)) that deliver better performance at a lower cost. We recommend always using the latest generation instances.

Avoid using T2 or t3.small instances for production domains because they can become unstable under sustained heavy load. t3.medium instances are an option for small production workloads (both as data nodes and as dedicated master nodes).

Use the latest Amazon EBS gp3 volumes

OpenSearch data nodes require low latency and high throughput storage to provide fast indexing and query. By using Amazon EBS gp3 volumes, you get higher baseline performance (IOPS and throughput) at a 9.6% lower cost than with the previously-offered Amazon EBS gp2 volume type. You can provision additional IOPS and throughput independent of volume size using gp3. These volumes are also more stable than previous generation volumes as they do not use burst credits. The gp3 volume type also doubles the per-data-node volume size limits of the gp2 volume type. With these larger volumes, you can reduce the cost of passive data by increasing the amount of storage per data node.

Use UltraWarm and cold storage for time-series log data

If you're using OpenSearch for log analytics, move your data to UltraWarm or cold storage to reduce costs. Use Index State Management (ISM) to migrate data between storage tiers and manage data retention.

[UltraWarm](#) ([./ultrawarm.html](#)) provides a cost-effective way to store large amounts of read-only data in OpenSearch Service. UltraWarm uses Amazon S3 for storage, which means that the

data is immutable and only one copy is needed. You only pay for storage that's equivalent to the size of the primary shards in your indexes. Latencies for UltraWarm queries grow with the amount of S3 data that's needed to service the query. After the data has been cached on the nodes, queries to UltraWarm indexes perform similar to queries to hot indexes.

[Cold storage \(./cold-storage.html\)](#) is also backed by S3. When you need to query cold data, you can selectively attach it to existing UltraWarm nodes. Cold data incurs the same managed storage cost as UltraWarm, but objects in cold storage don't consume UltraWarm node resources. Therefore, cold storage provides a significant amount of storage capacity without impacting UltraWarm node size or count.

UltraWarm becomes cost-effective when you have roughly 2.5 TiB of data in hot storage. Monitor your fill rate and plan to move indexes to UltraWarm before you reach that volume of data.

Review recommendations for Reserved Instances

Consider purchasing [Reserved Instances \(./ri.html\)](#) (RIs) after you have a good baseline on your performance and compute consumption. Discounts start at around 30% for no-upfront, 1-year reservations and can increase up to 50% for all-upfront, 3-year commitments.

After you observe stable operation for at least 14 days, review [Reserved Instance recommendations \(https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/ri-recommendations.html\)](#) in Cost Explorer. The **Amazon OpenSearch Service** heading displays specific RI purchase recommendations and projected savings.

© 2023, Amazon Web Services, Inc. or its affiliates. All rights reserved.

